Langmuir

Sep 06, 2021

Contents:

1 Installation **1**

2 Getting Started **3**

3 Specifying the plasma **5**
3.1 Quick computations of plasma parameters 6

4 Specifying the geometry **9**

5 Characteristic models **11**
5.1 Analytical theories 11
5.2 Normalization 12
5.3 Finite length models 13
5.4 Finite radius models 16

6 Examples **19**
6.1 Numerically solving for the floating potential 19
6.2 Fitting beta for a finite-length probe 20
6.3 Interactive finite-length current profile 21
6.4 Inferring plasma parameters from measurements 22

7 Citing Langmuir **35**

8 Bibliography **37**

9 Indices and tables **39**

Bibliography **41**

CHAPTER 1

Installation

Install from PyPI using `pip` (preferred method):

```
pip install langmuir
```

Or download the GitHub repository https://github.com/langmuirproject/langmuir.git and run:

```
python setup.py install
```

CHAPTER 2

Getting Started

The Langmuir library contains a collection of functions that compute the current collected by a conductor immersed in a plasma according to various models (characteristics). These functions take as arguments the probe geometry, for instance `Sphere`, and a plasma, where a plasma may consist of one or more `Species`.

As an example, consider a spherical Langmuir probe of radius $r = 1\,\mathrm{mm}$ immersed in a plasma with an electron density of $n = 10^{11}\,\mathrm{m}^{-3}$ and an electron temperature of $T = 1000\,\mathrm{K}$. The electron current collected by this probe when it has a voltage of $V = 4.5\,\mathrm{V}$ with respect to the background plasma is computed according to *orbital motion-limited* (OML) theory as follows:

```
>>> OML_current(Sphere(r=1e-3), Electron(n=1e11, T=1000), V=4.5)
-5.262656728335636e-07
```

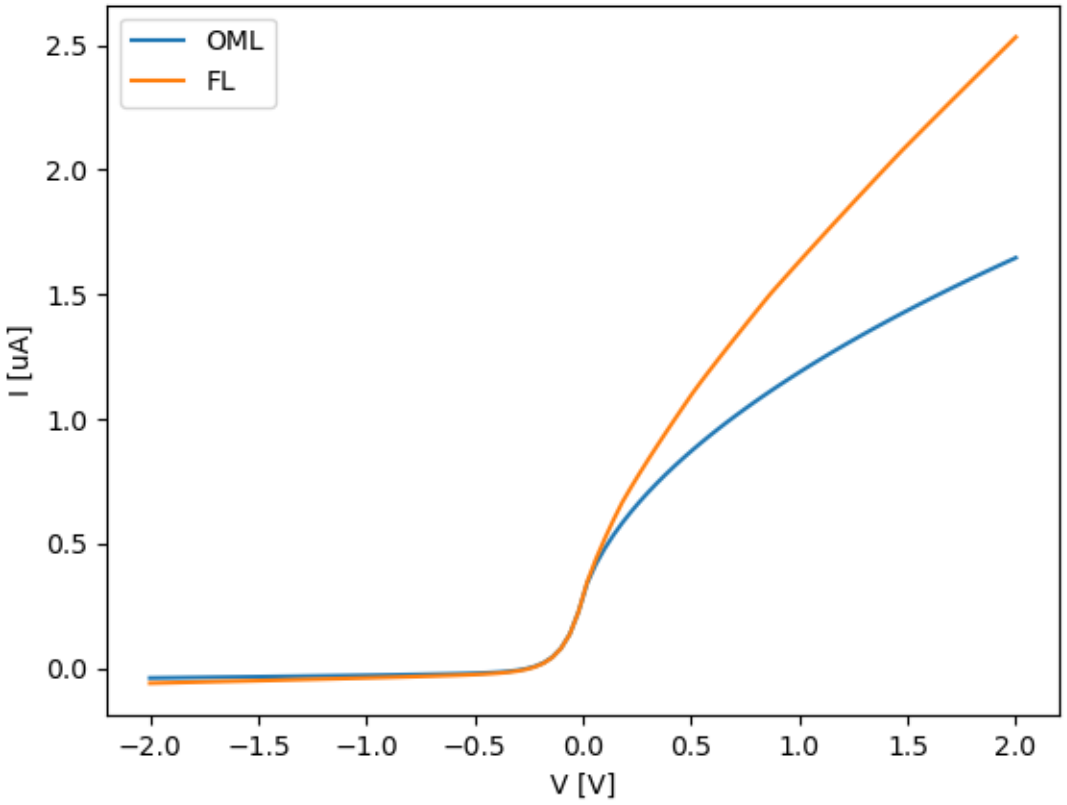
Let's consider a more complete example. Below we compare the current-voltage characteristics predicted by the OML theory and the *finite-length* (FL) model for a cylindrical probe with an ideal guard on one end.

```
from langmuir import *
import numpy as np
import matplotlib.pyplot as plt

plasma = [
    Electron(n=1e11, T=1000),
    Hydrogen(n=1e11, T=1000)
    ]

geometry = Cylinder(r=1e-3, l=60e-3, lguard=True)

V = np.linspace(-2, 2, 100)

I_OML = OML_current(geometry, plasma, V)
I_FL = finite_length_current(geometry, plasma, V)

plt.plot(V, -I_OML*1e6, label='OML')
plt.plot(V, -I_FL*1e6, label='FL')
plt.xlabel('V [V]')
```

(continues on next page)

(continued from previous page)

```
plt.ylabel('I [uA]')
plt.legend()
plt.show()
```



This example demonstrates that accounting for edge effects on a probe of finite length leads to a larger collected current. Also note that the characteristic correctly captures both the electron saturation, the electron retardation, and the ion saturation regions. Beware that the current collected by Langmuir probes (thus going into it) is usually negative. It is common practice, however, to invert it prior to plotting.

CHAPTER 3

Specifying the plasma

A species is specified using a range of keyword arguments upon initialization of a `Species` object, for instance:

```
Species(Z=1, amu=16, n=1e11, T=1000)
```

This is a singly charged ion ($Z = 1$) with a mass of $16\,\mathrm{AMU}$ (atomic mass units), which corresponds to the most abundant oxygen isotope. The density is $n = 10^{11}\,\mathrm{m}^{-3}$, and it is Maxwellian distributed with a temperature of $T = 1000\,\mathrm{K}$ and no drift-velocity. The full set of keywords is given below:

Quantity	Keyword	Units	Default value
Charge	`q`	Coulombs	elementary charge
	`Z`	elementary charges	
Mass	`m`	kilograms	electron mass
	`amu`	atomic mass units	
Density	`n`	partices per cubic meter	$10^{11}\,\mathrm{m}^{-3}$
Temperature/thermal speed	`T`	Kelvin	$1000\,\mathrm{K}$
	`eV`	electron-volts	
	`vth`	meters per second	
Spectral index kappa	`kappa`	dimensionless	∞ (`float('inf')`)
Spectral index alpha	`alpha`	dimensionless	0

Kappa, Cairns or Kappa-Cairns distributed particles are obtained by specifying `kappa`, `alpha`, or both, respectively. Maxwell, Kappa and Cairns distributions are all limiting cases of the Kappa-Cairns distribution [Darian].

For convenience, the following subclasses of `Species` exist:

- `Electron`
- `Proton`
- `Positron`
- `Antiproton`
- `Vogon`

- `Hydrogen`
- `Helium`
- `Lithium`
- and similar for the rest of the periodic table

The only difference from `Species` is that these have different default charge and mass to represent the named element. `Oxygen` for instance, defaults to singly charged oxygen of the most abundant isotope. Hence the example in the beginning of this section could also have been written:

```
Oxygen(n=1e11, T=1000)
```

Finally, a multi-species plasma is represented as a list of its constituents. A typical Oxygen plasma would for instance be:

```
plasma = [
    Electron(n=1e11, T=1000),
    Oxygen(n=1e11, T=1000)
    ]
```

3.1 Quick computations of plasma parameters

The Langmuir library is also suitable for doing quick computations of fundamental plasma parameters, since `Species` computes these upon initialization. One example is calculating the electron Debye length of a certain plasma:

```
>>> Electron(n=1e11, T=1000).debye
0.00690089806774598
```

The `Species` class defines the following useful members:

Member	Description
`debye`	The Debye length
`omega_p`	The angular plasma frequency
`freq_p`	The linear plasma frequency
`period_p`	The plasma period
`omega_c(B)`	The angular cyclotron frequency
`freq_c(B)`	The linear cyclotron frequency
`period_c(B)`	The cyclotron period
`larmor(B)`	The larmor radius

The latter four members are methods which take the magnitude of the magnetic flux density as an argument. In addition, every valid keyword argument of the constructor is also a valid member. This may conveniently be used for instance to convert a temperature from electron-volts to Kelvin:

```
>>> Species(eV=0.2).T
2320.9036243100163
```

In this case we only specified the input `eV`, since we know that temperature do not depend on density.

Finally, the total Debye length of a plasma consisting of multiple species can be obtained using the `debye()` function. For the oxygen plasma mentioned previously:

```
>>> debye(plasma)
0.004879671013271479
```

Note that Langmuir uses the correct expression for the Debye length also for general Kappa-Cairns distributed plasmas [Darian].

CHAPTER 4

Specifying the geometry

Langmuir supports three probe geometries, with self-descriptive names:

- `Plane(A)` represents a planar probe with surface area `A`.
- `Cylinder(r, l, lguard=0, rguard=0)` represents a cylindrical probe of radius `r` and length `l`. The optional arguments `lguard` and `rguard` may be used to specify the length of the left and right guards, respectively. Setting them to `float('inf')` or `True` means that there is an ideal guard.
- `Sphere(r)` represents a spherical probe of radius `r`.

All dimensions are in SI units. Not all models may be able to support all geometries, and only the finite-length model makes use of the guard feature.

CHAPTER 5

Characteristic models

Langmuir comes with several models for the characteristic $I(V)$ of Langmuir probes. Each model is represented by a function which, in addition to the voltage `V`, take a `geometry` and a `species` argument. The models have similar signatures and can often be interchanged. `geometry` is one of the previously mentioned probe geometries (`Plane`, `Cylinder` or `Sphere`), and the `species` argument is either a single `Species` object, or a list of `Species`. If it is a single species, the return value is the current collected from that species alone. This may be sufficient in regimes where the current is dominated by the collection of one species. If the `species` argument it is a list of species, it is the sum of the currents from all species (there is currently no model which accounts for non-linear coupling between species). If both ions and electrons are included, the characteristics captures both the electron saturation, electron retardation and ion saturation regimes. The model functions also have arguments `eta` and `normalization` which will be covered in *Normalization*.

5.1 Analytical theories

5.1.1 The orbital motion-limited (OML) theory

The *orbital motion-limited* (OML) theory is the de-facto standard theory for current collection of a Langmuir probe, and is implemented as the following function:

```
OML_current(geometry, species, V=None, eta=None, normalization=None)
```

Assuming a small probe radius compared to the Debye shielding distance, i.e., the *thick sheath approximation*, the current collected by a probe in a plasma can be calculated based on orbital trajectories within the sheath. The derivations of the implemented analytical expressions assumes:

1. a collisionless plasma
2. a non-drifting Kappa-Cairns-distributed plasma
3. a non-magnetized plasma
4. small probe radius (if spherical or cylindrical) compared to the Debye length
5. large probe length (if cylindrical), or large extent (if planar) compared to the Debye length

It is worth remarking that the Maxwell, Kappa and Cairns distributions are special cases of the Kappa-Cairns distributions and are therefore also covered [Darian]. Moreover, no assumptions are made on the voltage range.

Radii up to 0.2 Debye lengths (for spherical probes) or 1.0 Debye lengths (for cylindrical probes) typically satisfy the small radius criterion well [Laframboise]. Cylindrical probes need to be very long, however, to satisfy the long probe criterion [Marholm]. Other models in Langmuir overcome some of the limitations of the OML theory.

For more information on OML theory, see [MottSmith] for the original derivation, or [Darian] for the more general derivation with Kappa-Cairns distribution.

5.1.2 Thermal currents and the sheath-limited (SL) theory

A probe at zero voltage with respect to the background plasma do not accelerate particles, but nevertheless collects a current due to the random thermal motion of particles through the surface of the probe. This is implemented in the following function:

```
thermal_current(geometry, species, normalization=None)
```

This coincides with the OML theory not only for zero voltage, but also for planar probes. This can be understood from the fact that for planar probes, the effective collection area (the sheath boundary) do not increase as the voltage increases and the sheath thickness increases, but remains of the same area.

The *sheath-limited* (SL) theory makes the same assumptions as the OML theory, except that the probe radius is large compared to the Debye length (*thin sheath approximation*). With these approximations, the current collected is a/r times the thermal current, where a is the distance from the center of the probe to the edge (in some sense) of the sheath surrounding the probe, and r is the probe radius. For spheres and cylinders an increase in voltage causes an increase in a and hence in collected current. Nevertheless, for very large probes, a/r approches unity, and *thermal_current* may be used as an approximation. In this case, the probe can be understod to be locally flat.

5.2 Normalization

The characteristics of Langmuir probes do not depend on every conceivable parameter such as density, temperature, probe length, etc. independently, but instead upon a smaller set of non-dimensional groups (π-groups) of such parameters [Laframboise], [Marholm]. One such group is the normalized voltage:

$$\eta = -\frac{qV}{kT}$$

where q and T are the charge and temperature of the collected species, V is the voltage, and k is Boltzmann's constant (for $\eta > 0$ the species is attracted, and for $\eta < 0$ it is repelled). It may therefore be of interest to study an entire class of problems with the same normalized parameters instead of a specific case. In Langmuir one may specify normalized voltages by using the argument `eta` instead of `V` in models such as `OML_current`.

Similarly, a normalized current I/I_0 may be defined, where I is the collected current and I_0 is a characteristic current. The models in Langmuir may return currents normalized by one of several possible characteristic currents depending on the value of the `normalization` argument:

- `'th'`: Normalized by the *thermal current*. Often the most natural choice.
- `'oml'`: Normalized by the current according to *the OML theory*. This is useful for comparing other models with OML theory.
- `'thmax'`: Normalized by the thermal current of a Maxwellian plasma regardless of what the distribution actually is. This is the normalization used in [Darian].

Finally, all lengths are normalized by the Debye length λ_D. Below is a complete example of obtaining the normalized current for a cylindrical probe of radius $0.2\lambda_D$ and length $10\lambda_D$ with $\eta = 10$:

```
>>> sp = Species()
>>> geometry = Cylinder(r=0.2*sp.debye, l=10*sp.debye)
>>> OML_current(geometry, sp, eta=10, normalization='th')
3.7388259506315147
```

Since only the non-dimensional groups determine the normalized collected current, we do not care about the exact parameters of the species, but leave them at the default. Note that in this case it actually does not matter what the probe size is, because the thermal current depends on the probe size in the same way as the current predicted by OML theory. This may differ for other models, however.

Note that the non-dimensional groups are specific to each species, e.g., the voltage normalized with respect to electrons is not the same as with respect to ions. If a multi-species plasma is specified, the normalization used will be with respect to the first species in the list. E.g., if electrons are the first element in the list, $\eta = eV/kT_e$ where e is the elementary charge and T_e is the electron temperature.

5.3 Finite length models

The two finite-length models are based on theoretical scaling laws and numerical simulations, and accounts for edge effects on cylindrical probes. They assume a collisionless, non-magnetized and non-drifting Maxwellian plasmas, as well as a small probe radius. However, they can account for probes of arbitrary length, all the way down to below the Debye length (at which point the probes behave as if they were spherical). The models are valid for normalized voltages up to 100. See [Marholm] for a full description.

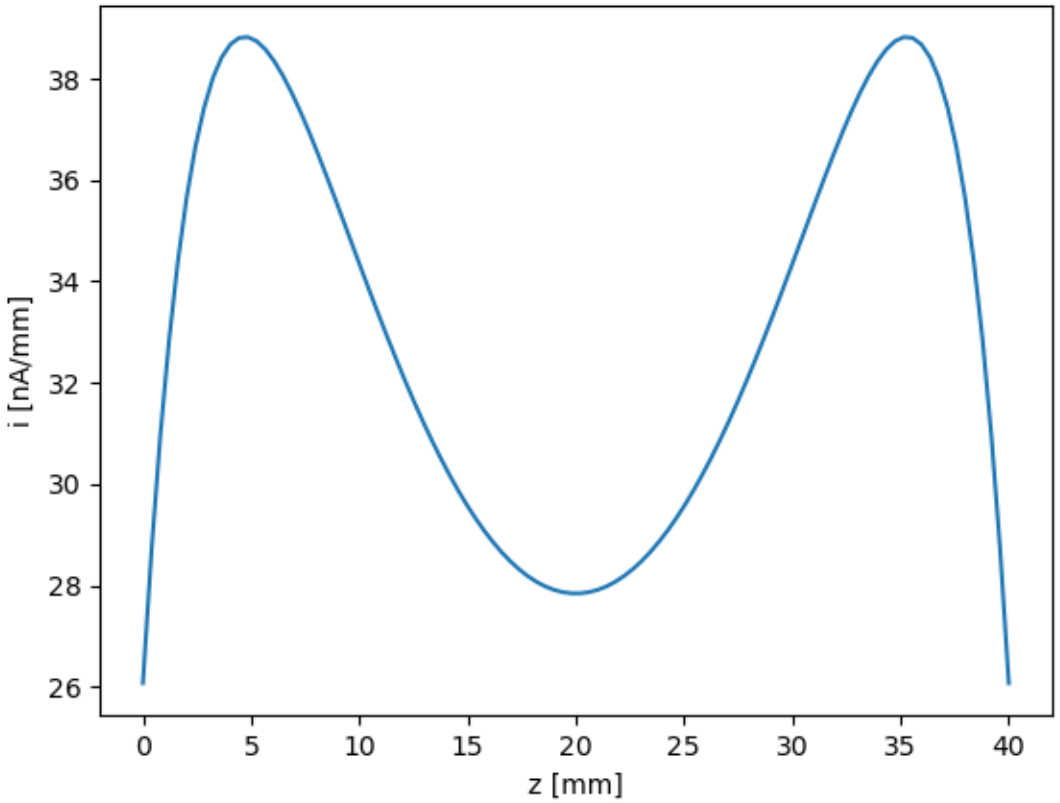
5.3.1 Current per unit length

The first model describes not the total collected current, but the current $i(z)$ per unit length, as a function of the position z on the probe, and has the following signature:

```
finite_length_current_density(geometry, species, V=None, eta=None, z=None, zeta=None,␣
↪normalization=None)
```

This model has an additional argument `z`, which is the position on the probe, starting at zero. Alternatively, one may use the normalized position `zeta`, defined as $\zeta = z/\lambda_D$. Below is an example for a 40-mm probe with both ends free:

```
from langmuir import *
import numpy as np
import matplotlib.pyplot as plt

elec = Electron(n=1e11, T=1000)
l = 40e-3
z = np.linspace(0,l,100)

geo = Cylinder(r=0.25e-3, l=l)
i = finite_length_current_density(geo, elec, V=5, z=z)

plt.plot(z*1e3, -i*1e6)
plt.xlabel('z [mm]')
plt.ylabel('i [nA/mm]')
plt.show()
```



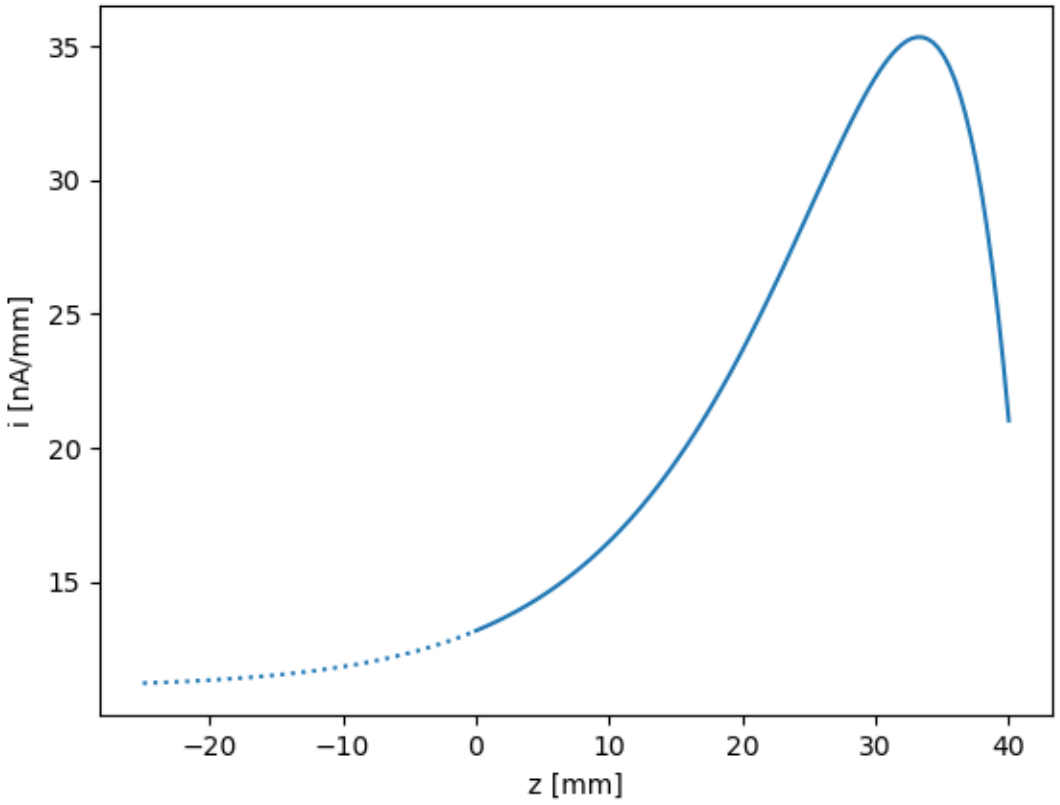
The end effects from the two ends are quite severe, and they even overlap. In reality, however, one end of the probe must necessarily be attached to something, and this is usually a guard. Ideally, a guard is an infinite cylindrical extension of the probe at the same voltage as the probe, but electrically insulated from it, such that current collected by the guard is not included in the measurements. This is supposed to remove end effects from this end of the probe. It is possible to enable ideal guards by setting either `lguard` (left guard) or `rguard` (right guard) in `Cylinder` to `True` or `float('inf')`. In the modified example below, we include a guard to the left of the probe, and we also plot the current collected by the nearest part of the guard:

```
from langmuir import *
import numpy as np
import matplotlib.pyplot as plt

elec = Electron(n=1e11, T=1000)

l_probe = 40e-3
l_guard = 25e-3
z_probe = np.linspace(0, l_probe, 100)
z_guard = np.linspace(-l_guard, 0, 100)

geo = Cylinder(r=0.25e-3, l=l_probe, lguard=True)
i_probe = finite_length_current_density(geo, elec, V=5, z=z_probe)
i_guard = finite_length_current_density(geo, elec, V=5, z=z_guard)

plt.plot(z_probe*1e3, -i_probe*1e6)
plt.plot(z_guard*1e3, -i_guard*1e6, ':C0')
```

(continues on next page)

(continued from previous page)

```
plt.xlabel('z [mm]')
plt.ylabel('i [nA/mm]')
plt.show()
```



If finite values are used for `lguard` or `rguard`, the end of the guard pointing away from the probe will assume end effects similar to that of a free cylindrical end, and these end effects may extend into the probe if the guard is short. In reality a guard will often be mounted on a spacecraft or some arbitrary body, which will lead to a different end effect than that of a free end. Due to the arbitraryness of this end effect, it cannot be predicted by the finite-length model. For this reason, and because the model is believed to be more accurate for large probe plus guard lengths (see [Marholm]), it is advisable to specify ideal guards in most cases. For the same reasons, it is also advisable that instruments be designed with long guards (long probes are in fact less important, since their non-ideal effects can be predicted).

5.3.2 Total collected current

The second model describes the total collected current, and is merely the integral of the current per unit length. Its signature is as follows:

```
finite_length_current(geometry, species, V=None, eta=None, normalization=None,␣
↪interpolate='I')
```

See example of use in *Getting Started*.

5.3.3 Behind the curtains

This section describes implementation details beyond what is mentioned in [Marholm]. The current per unit length in the finite-length model (`finite_length_current_density`) is given by the following expression [Marholm]:

$$i(z) = i_{\mathrm{OML}}\, g(\zeta; \lambda, \eta)$$

where i_{OML} is the current per unit length according to the OML theory, and

$$\zeta = \frac{z}{\lambda_D}, \quad \lambda = \frac{l}{\lambda_D}, \quad \eta = -\frac{qV}{kT}$$

are the non-dimensional groups. z is the position on the probe, which spans from 0 to l. Moreover,

$$g(\zeta) = C(1 + h(\zeta) + h(\lambda - \zeta))$$
$$h(\zeta) = A(\zeta - \delta + \alpha^{-1}) \exp(-\alpha\zeta)$$

where A, C, α and δ are coefficients that are known for fixed pairs (λ, η) on a regular grid.

For probes with guards, the total normalized length is taken to be $\lambda = \lambda_l + \lambda_p + \lambda_r$, where λ_l and λ_r are the normalized lengths of the left and right guards, respectively, whereas λ_p is the normalized length of the actual probe. The end effects still have the same shape, but are (partially) absorbed by the guards. However, in Langmuir the origin is shifted, such that the `zeta` parameter is actually $\zeta_p = \zeta - \lambda_l$. With this shift, the probe strecthes from $\zeta_p = 0$ to $\zeta_p = \lambda_p$, and the guards extend beyond this range on either side. The g-function may be re-written as a function of ζ_p:

$$g(\zeta_p) = C(1 + h(\lambda_l + \zeta_p) + h(\lambda_p + \lambda_r - \zeta_p))$$

The total current collected by the probe (`finite_length_current`) is obtained by integrating $i(z)$ over the probe, excluding the guards,

$$I = i_{\mathrm{OML}} \lambda_D G$$
$$G = \int_0^{\lambda_p} g(\zeta_p)\, \mathrm{d}\zeta_p = C(\lambda_p + H(\lambda_p + \lambda_l) + H(\lambda_p + \lambda_r) - H(\lambda_l) - H(\lambda_r))$$
$$H(\zeta) = A \frac{\alpha(\delta - \zeta) - 2}{\alpha^2} \exp(-\alpha\zeta)$$

The shift of origin is not only convenient, but eliminates the need to subtract λ_l from $\lambda_l + \lambda_p$, which could lead to issues of numerical accuracy when λ_l is large, or even tends to infinity. The above expressions, on the other hand, have no such differences, and behave gracefully as the guard lengths increases to infinity.

For parameters (λ, η) *not* on the regular grid in parameter space, interpolation is necessary. One alternative is to linearly interpolate the coefficients A, C, α and δ. However, as described in [Marholm] this may not behave well for short probe plus guard lengths. For the total collected currents it is also possible to calculate I for the four nearest parameters (λ, η) on the grid, and interpolate I linearly. This gives a smoother result, since I then varies linearly with (λ, η). This is now the default behaviour, but I may still be derived from interpolated coefficients by setting `interpolate='g'` in `finite_length_current`.

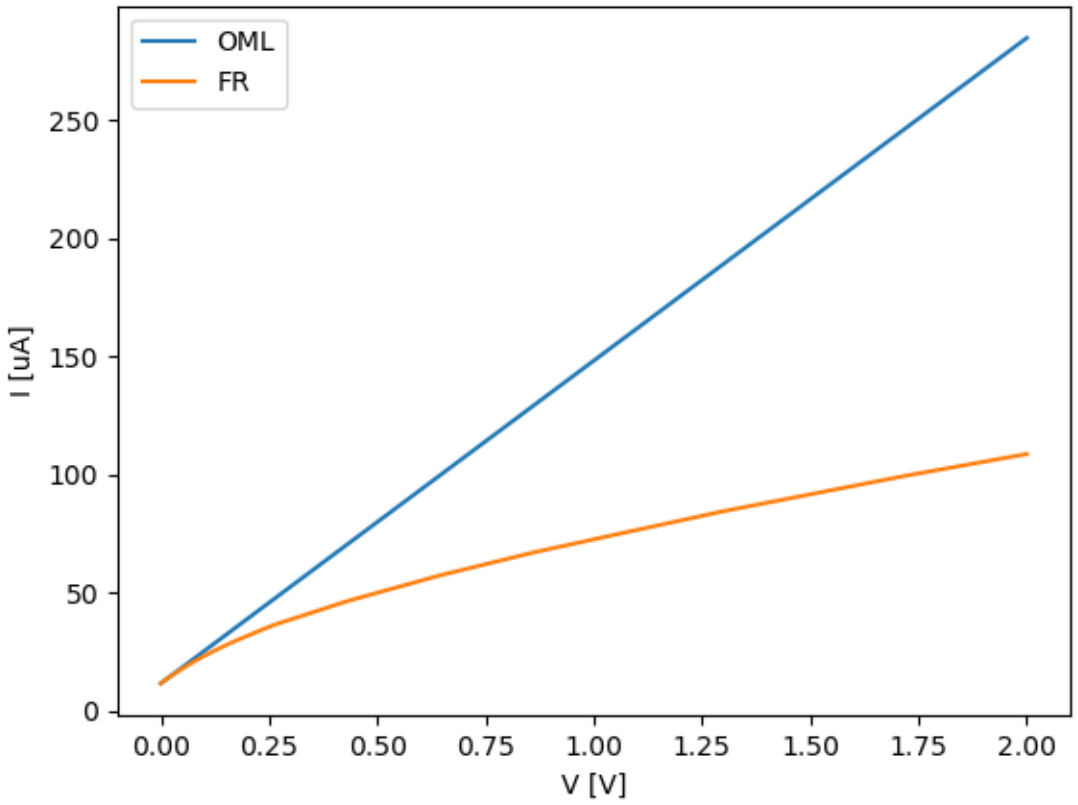
5.4 Finite radius models

Finite radius models relax the restriction on the OML theory that the radius should be small, and may be accessed through the following function:

```
finite_radius_current(geometry, species, V=None, eta=None, table='laframboise-darian-
↪marholm', normalization=None)
```

Example of use:

```
from langmuir import *
import numpy as np
import matplotlib.pyplot as plt

plasma = Electron(n=1e11, T=1000)
geometry = Sphere(r=5*debye(plasma))

V = np.linspace(0, 2, 100)

I_OML = OML_current(geometry, plasma, V)
I_FR = finite_radius_current(geometry, plasma, V)

plt.plot(V, -I_OML*1e6, label='OML')
plt.plot(V, -I_FR*1e6, label='FR')
plt.xlabel('V [V]')
plt.ylabel('I [uA]')
plt.legend()
plt.show()
```



The model is built from two sets of tabulated numerical results, which can be found in [Laframboise] and [Darian]. The former is more accurate but covers only Maxwellian plasmas, whereas the latter covers the more general Kappa-Cairns distributions. By default interpolation between tabulated values will use the most accurate values. The model is valid for $0 \leq \eta \leq 25$, $\kappa \geq 4$ and $\alpha \leq 0.2$. Probe radii extends up to 10 Debye lengths, or even 100 Debye lengths as the distribution approaches Maxwellian.

CHAPTER 6

Examples

6.1 Numerically solving for the floating potential

In this example we consider how to numerically obtain the floating potential by means of a root solver. We shall consider a sphere situated in a hydrogen plasma, which is known to have a floating potential of $-2.5kT/e$, where k is Boltzmann's constant, T is the temperature, and e is the elementary charge [Whipple]. The code for the example is given below:

```
from langmuir import *
from scipy.constants import value as constants
from scipy.optimize import root_scalar

n=1e11
T=1000
e = constants('elementary charge')
k = constants('Boltzmann constant')

plasma = [Electron(n=n, T=T),
          Hydrogen(n=n, T=T)]

geometry = Sphere(r=0.2*debye(plasma))

def res(V):
    return OML_current(geometry, plasma, V)

sol = root_scalar(res, x0=-3, x1=0)

print(sol.root)
print(-2.5*k*T/e)
```

The requirement for a steady floating potential is that the net current into the object, by all species, is zero. Otherwise the potential would increase or decrease. Thus, in our case we seek to find the root of the characteristic, which we take as `OML_current` in the example above. The function `root_scalar` from SciPy iterates on the value `V` of `res` within the range $(-3, 0)$, until it returns a value that is sufficiently close to zero, i.e., the root of `OML_current`. The

script returns:

```
-0.2157892685234552
-0.21543333155362945
```

and hence is in excellent agreement with previous results.

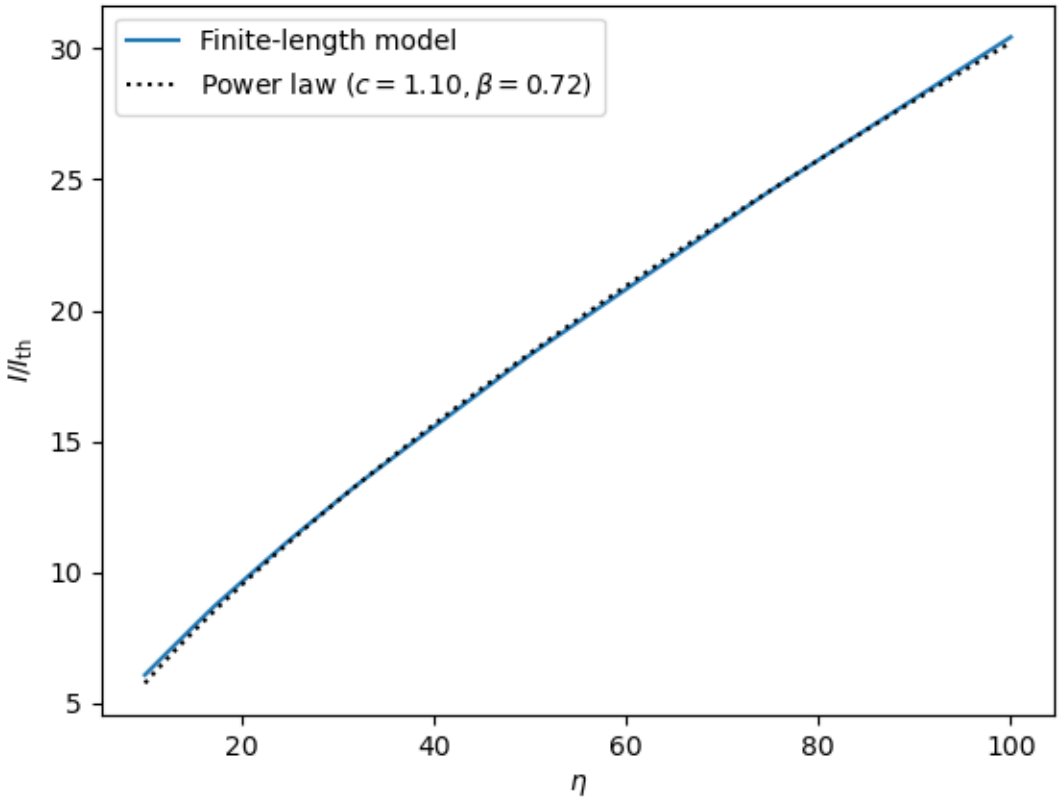
6.2 Fitting beta for a finite-length probe

For large voltages, $\eta = qV/kT \gg 1$, the normalized current collected from the attracted species according to the OML theory may be written as power law:

$$\frac{I}{I_{\mathrm{th}}} \approx c\eta^{\beta}$$

where β is 0, 0.5 or 1 for a plane, cylinder or sphere, respectively. It is customary to use this same expression for probes of finite length as well, but with β varying between 0.5 and 1. This example demonstrates how to use a standard curve fitting algorithm to find β for a cylindrical probe of ten Debye lengths:

```
from langmuir import *
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit

elec = Electron()
eta = np.linspace(10,100,100)
geo = Cylinder(r=0.1*elec.debye, l=10*elec.debye)
I = finite_length_current(geo, elec, eta=eta, normalization='th')

def power_law(eta, c, beta):
    return c*eta**beta

popt, pcov = curve_fit(power_law, eta, I)

plt.plot(eta, I, label='Finite-length model')
plt.plot(eta, power_law(eta, *popt), ':k',
         label=r'Power law ($c={:.2f}$, \beta={:.2f})$'.format(*popt))

plt.xlabel(r'$\eta$')
plt.ylabel(r'$I/I_\mathrm{th}$')
plt.legend()
plt.show()
```



In the code we start by evaluating the charactersitic at 100 points along $\eta \in [10, 100)$. Note that we keep the normalized voltage always above 10 such that the large voltage approximation is satisfied. We then define the function we want to fit the characteristic to, `power_law`, with the input as the first argument, followed by an arbitrary number of fitting coefficients (in our case two). The SciPy function `curve_fit` makes a best fit of these coefficients, returned in the tuple `popt`. With these coefficients, the fit (dotted line) is in excellent agreement with the actual characteristic (solid line), and $\beta = 0.72$.

6.3 Interactive finite-length current profile

In this example the current per unit length is plotted along a cylindrical probe, allowing the user to interactively change then length and voltage of the probe. All quantities are normalized.

```
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider
from langmuir import *
from scipy.special import erf

normalization='OML'

elec = Electron()
geo = Cylinder(r=0.1*elec.debye, l=20*elec.debye)
z = np.linspace(0, 20, 1000)
```

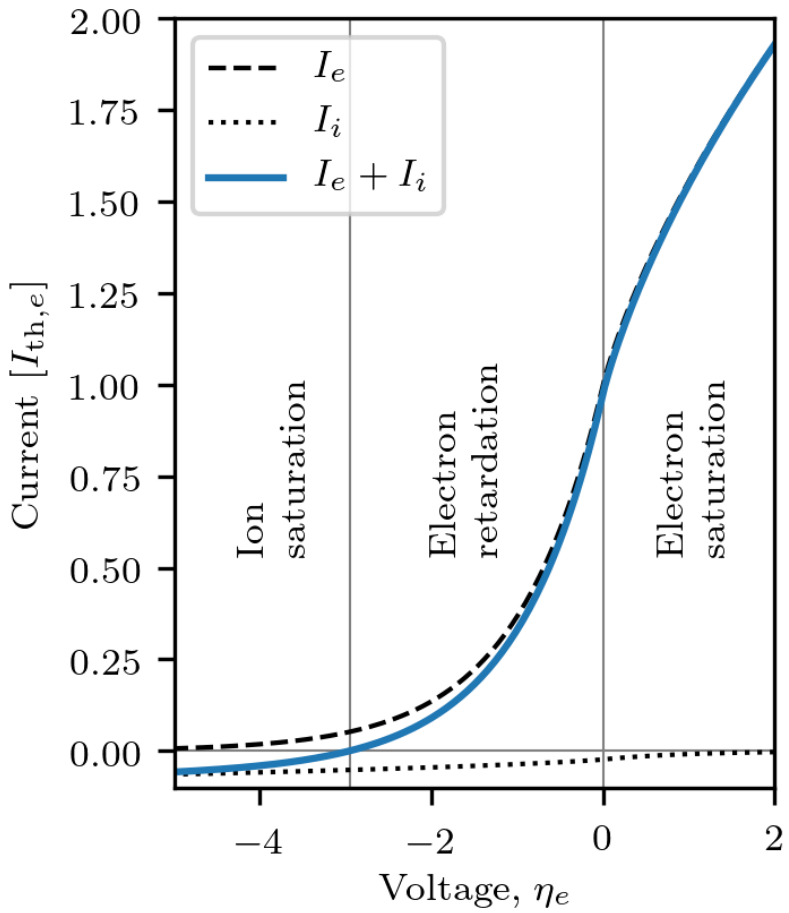
(continues on next page)

(continued from previous page)

```
I = finite_length_current_density(geo, elec, zeta=z, eta=25,␣
↪normalization=normalization)

len_min = 1
len_max = 40
eta_min = -10
eta_max = 100

fig, ax = plt.subplots()
plt.subplots_adjust(bottom=0.25)
line, = plt.plot(z, I)
if normalization=='th':
    ax.axis([-5, len_max+5, 0.5, 20])
else:
    ax.axis([-5, len_max+5, 0.5, 4.5])
ax.grid()
plt.xlabel(r'$z/\lambda_D$')
plt.ylabel(r'$I/I_\mathrm{OML}$')

ax_len = plt.axes([0.1, 0.10, 0.8, 0.03])
ax_eta = plt.axes([0.1, 0.05, 0.8, 0.03])
sl_len = Slider(ax_len, r'$l/\lambda_D$', 1, len_max, valinit=20)
sl_eta = Slider(ax_eta, r'$\eta$', eta_min, eta_max, valinit=25)

def update(val):
    eta = sl_eta.val
    l = sl_len.val
    z = np.linspace(0, l, 1000)
    geo = Cylinder(r=0.1*elec.debye, l=l*elec.debye)
    I = finite_length_current_density(geo, elec, zeta=z, eta=eta,␣
↪normalization=normalization)
    line.set_ydata(I)
    line.set_xdata(z)

sl_len.on_changed(update)
sl_eta.on_changed(update)

plt.show()
```

6.4 Inferring plasma parameters from measurements

The purpose of Langmuir probes is to measure plasma parameters, such as the electron and ion densities, and the electron temperature. The traditional techniques rely on OML theory, which predicts that the characteristic behaves differently in different regions of the probe voltage. For voltages between the floating potential and the background plasma potential (here taken to be zero), for instance, the ion current can be neglected and the OML theory then predicts a slope $\mathrm{d}(\ln I)/\mathrm{d}V$ depending only on the electron temperature, as well as some physical constants. Doing a voltage sweep across this region therefore allows the determination of the electron temperature. Further on, in the electron (ion) saturation region, the ions (electrons) can be neglected, and the analytical expressions of the remaining part allows determination of the electron (ion) density, once the electron temperature is known [Marholm2], [Bekkeng], [MottSmith], [Bittencourt].



Another technique is that of Jacobsen and Bekkeng for the multineedle Langmuir probe (m-NLP) instrument [Jacobsen]. The m-NLP instrument consists of at least two (typically four) cylindrical Langmuir probes biased at different fixed positive voltages with respect to a spacecraft. The OML theory predicts that the slope $\mathrm{d}I^2/\mathrm{d}V$ depends only on the electron density, except for known physical constants. The m-NLP instrument thus allows inferring the electron density without sweeping the voltage, which gives the m-NLP instrument faster sampling times and thus higher spatial resolution while spaceborn than swept Langmuir probes. A fast implementation of this density inference technique is readily available in Langmuir. Given an $N \times 4$ array `I`, where each row corresponds to the currents measured at some time instant by probes biased at, say, 2, 3, 4, and 5 volts with respect to some common reference voltage, the densities can be inferred as follows:

```
n = jacobsen_density(Cylinder(r=0.255e-3, l=25e-3), [2,3,4,5], I)
```

The main problem with these approaches is that they rely upon specific analytic expressions for the characteristic, which may not hold for non-ideal cases (finite length, finite radius, collisional or non-Maxwellian plasmas). Another problem is in identifying the different regions. In spaceborn Langmuir probes, for instance, the probe voltage is only known with respect to the spacecraft, and not with respect to the background plasma.

6.4.1 A general formulation

In relation to the Langmuir software we take a more general point of view [Marholm2]. Consider that a set of currents $\{\hat{I}_p\}_{p=1}^N$ have been measured in a plasma. The currents may for instance be the currents corresponding to different voltages of a swept Langmuir probe, or it may be the currents collected by different probes, such as in the m-NLP instrument. For the sake of generality, we allow each measurement to obey a different characteristic function, which

we denote $I_p(V_p; \mathbf{P})$. V_p is the probe voltage at which the current was measured, and $\mathbf{P}$ is a vector of other parameters that the characteristics depend upon, such as electron temperature and density. The probe voltage is often not known with respect to the background plasma, but instead with respect to a common reference voltage V_0. For spaceborn instruments this is the spacecraft floating potential. With this, we may write $V_p = V_0 + V_{0p}$, and the measurements form the following system of equations:

$$\hat{I}_p = I_p(V_0 + V_{0p}; \mathbf{P}), \quad p = 1, ..., N$$

This set of equations may be solved for the unknowns $(V_0, \mathbf{P})$ by any suitable numerical method (insofar as it is well-posed), and there is a wide range of free software available depending on how this system is to be solved. However, it requires programmatic access to the characteristics $I_p(V_p; \mathbf{P})$. Computing the currents for given physical parameters may be considered the *forward problem*, and it is a prerequisite for solving the *inverse problem*, namely inferring physical parameters from measured currents. Langmuir focuses on the forward problem. In the following, however, we give a few examples of attacking the inverse problem.

6.4.2 Synthetic data

For experimentation, the following function can be used to generate a test set of synthetic currents:

```
generate_synthetic_data(geometry,
                        V,
                        model=finite_length_current,
                        V0=None,
                        alt_range=(100,500),
                        noise=1e-5)
```

The currents are synthesized assuming densities from IRI [IRI] and temperatures from MSIS [MSIS] for 45 degrees latitude, 0 degrees longitude, and altitudes within the range given by `alt_range` at local noon. The advantage of synthetic data is that since the ground truth is known, it can be used to measure the accuracy of the inversion. Beware, however, that `generate_synthetic_data` necessarily must use some model to generate the currents (given by `model`), and the net accuracy of the final inferred parameters will depend both on the accuracy of the inversion *and* of the forward model for the problem at hand.

`generate_synthetic_data` will also add noise proportional to the square root of the signal strength and the factor `noise` for a representative signal-to-noise ratio [Ikezi], as well as generate a synthetic floating potential. Alternatively the floating potential can be preset by the argument `V0`. For a given `geometry` and array of bias voltages `V`, the function returns a dictionary with the following arrays:

- `alt`: Altitude of the data samples [km]
- `I`: Synthetic current measurements [A], one column for each bias voltage
- `V0`: Ground truth, floating potential [V]
- `Te`: Ground truth, electron temperature [K]
- `Ti`: Ground truth, ion temperature [K]
- `ne`: Ground truth, electron density [1/m^3]
- `nO+`: Ground truth, density of oxygen ions [1/m^3]
- `nO2+`: Ground truth, density of doubly charged oxygen ions [1/m^3]
- `nNO+`: Ground truth, density of nitrosonium ions [1/m^3]
- `nH+`: Ground truth, density of hydrogen ions [1/m^3]

6.4.3 Inversion by least-squares curve fitting

In this section we consider how to infer electron density and floating potential from current measurements by an iterative non-linear least squares curve fitting algorithm. Least-squares fitting works by minimizing a sum of squared residuals r_p,

$$R = \sum_p r_p^2$$

with respect to the fitting coefficients which in our case is (n_e, V_0). The residuals, we take as

$$r_p = (I_p(V_0 + V_{0p}; \mathbf{P}) - \hat{I}_p)$$

such that a perfect solution yields $R = 0$.

We shall assume a 4-NLP instrument with bias voltages V_{0p} of 2, 3, 4 and 5 volts with respect to the unknown floating potential V_0. There will then be four residuals per time sample, and in order to use SciPy's *least_squares* function we need to define a function *residual* returning a vector of these four residuals when given an approximation of the coefficients $x = (n_e, V_0)$ as its first argument.

To begin with, it is advisibale to verify the method on a simpler case. We therefore assume that OML theory is a perfect representation of reality, and use it to generate synthetic ground truth data along with `generate_synthetic_data`. We then do the least-squares fit of the synthesized currents to the OML characteristic (I_p is given by `OML_current`), and verify that we are able to make predictions close to the ground truth. The code reads as follows:

```
from langmuir import *
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import least_squares
from tqdm import tqdm

geometry = Cylinder(r=0.255e-3, l=25e-3, lguard=True)
V = np.array([2,3,4,5])

model_truth = OML_current
model_pred = OML_current

data = generate_synthetic_data(geometry, V, model=model_truth)

alt = data['alt']
ne_jacobsen = jacobsen_density(geometry, V, data['I'])
ne_fit = np.zeros_like(alt)
V0_fit = np.zeros_like(alt)

def residual(x, I, T):
    n, V0 = x
    return (model_pred(geometry, Electron(n=n, T=T), V+V0) - I)*1e6

x0 = [10e10, -0.3]
x_scale = [1e10, 1]
for i in tqdm(range(len(alt))):
    I = data['I'][i]
    T = data['Te'][i]
    T = 2000
    res = least_squares(residual, x0, args=(I,T), x_scale=x_scale)
    ne_fit[i], V0_fit[i] = res.x
    x0 = res.x
```

(continues on next page)

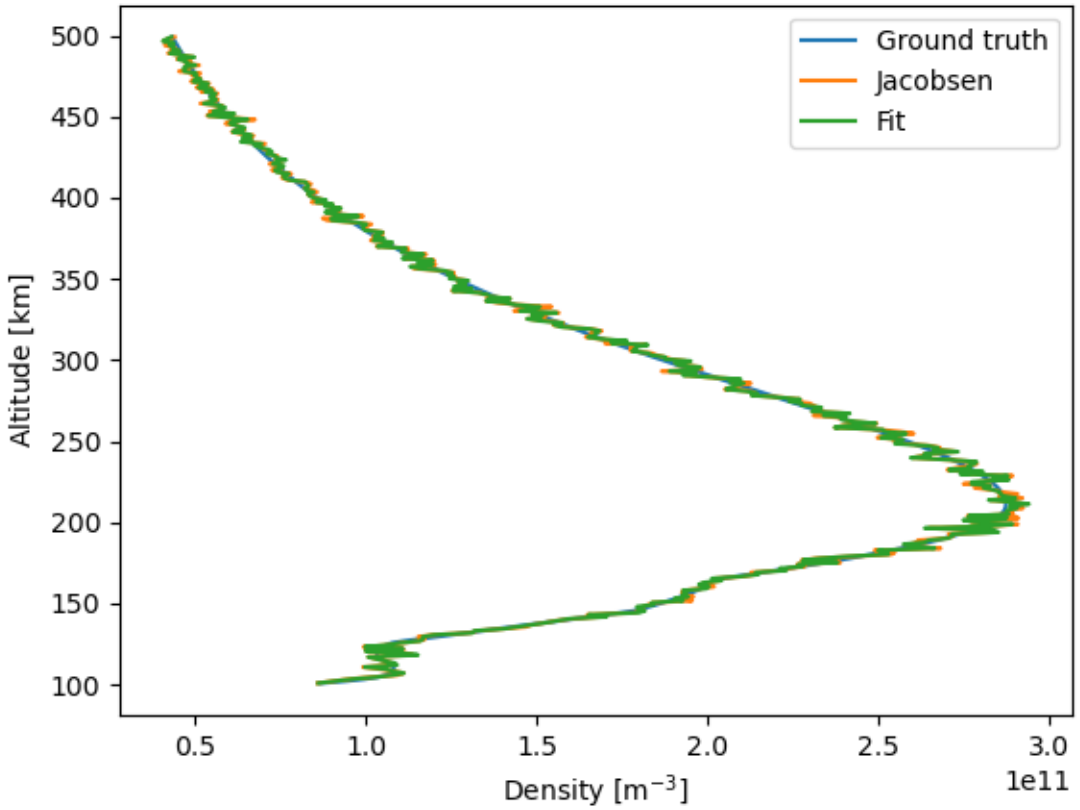
(continued from previous page)

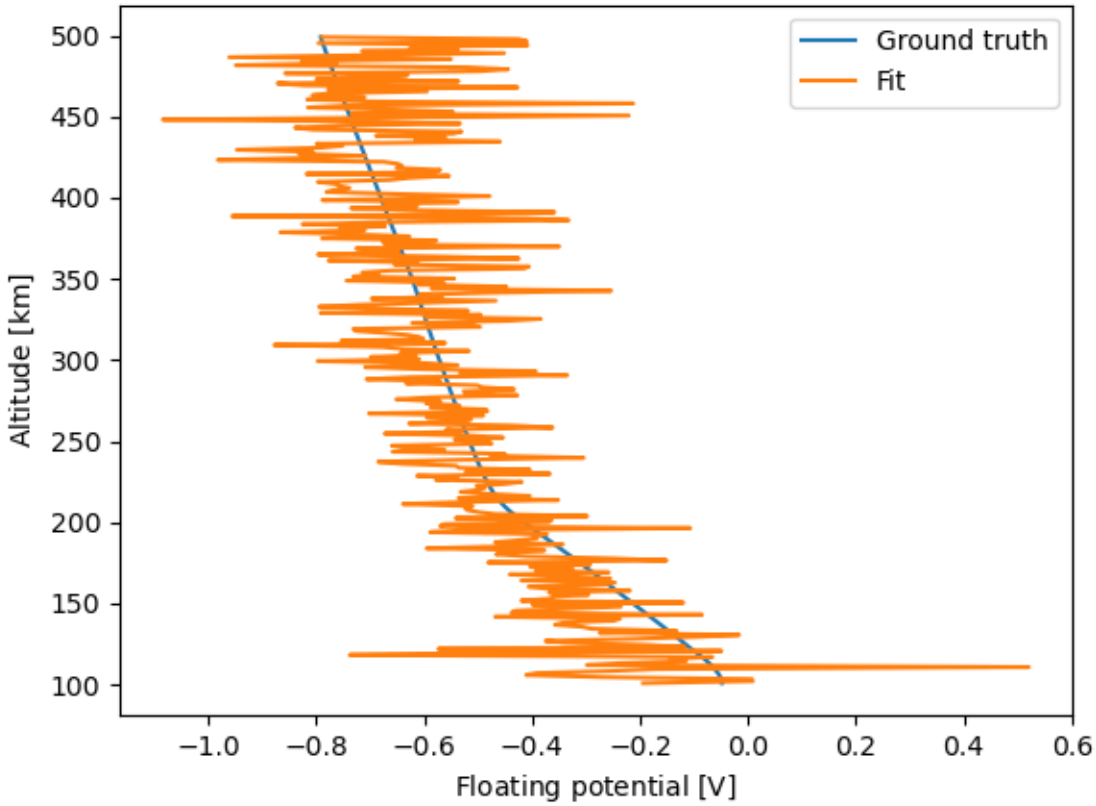
```
plot = plt.plot

plt.figure()
plot(data['ne'], alt, label='Ground truth')
plot(ne_jacobsen, alt, label='Jacobsen')
plot(ne_fit, alt, label='Fit')
plt.xlabel('Density $[\mathrm{m}^{-3}]$')
plt.ylabel('Altitude $[\mathrm{km}]$')
plt.legend()

plt.figure()
plot(data['V0'], alt, label='Ground truth')
plot(V0_fit, alt, label='Fit')
plt.xlabel('Floating potential $[\mathrm{V}]$')
plt.ylabel('Altitude $[\mathrm{km}]$')
plt.legend()

plt.show()
```

The loop carries out the fit over each sample, and to increase our chance of success and possibly reduce the number of iterations required, we try to give it an initial guess `x0` that is as close as possible to the true value as possible. For the first sample this is a pre-set value, and for subsequent samples we use the previous solution.

Because numerical algorithms often work best for numbers close to unity, we also scale the coefficients (n_e, V_0). This is conveniently handled by `least_squares` itself, which accepts an argument `x_scale` with numbers of typical magnitude. The residuals are in the order of microampére's and also needs to be scaled. However, although `least_squares` has an argument `f_scale` for the residuals, it is not always in use. We therefore multiply the residuals by `1e6` ourselves.

The residual function may also accept arguments that are not part of the optimization, in our case the four currents stored in the vector `I` and the electron temperature `T`. These are passed through to `residual` using the `args` argument of `least_squares`. Although the electron temperature is technically an unknown parameter (an input to the forward model), it is hard to infer it because the characteristic only depends very weakly on it (see [Hoang], [Barjatya], [Marholm2]). We therefore specify it manually. This could just be a representative number (2000K in our case), or it could be a number from another instrument or model. For our example with synthetic data it is also possible to use the ground truth directly, by replacing the line `T = 2000` with `T = data['Te'][i]`. Execution results in the following plots:


Ground truth
Jacobsen
Fit
Altitude [km]
500
450
400
350
300
250
200
150
100
0.5
1.0
1.5
2.0
2.5
3.0
1e11
Density [m^{-3}]

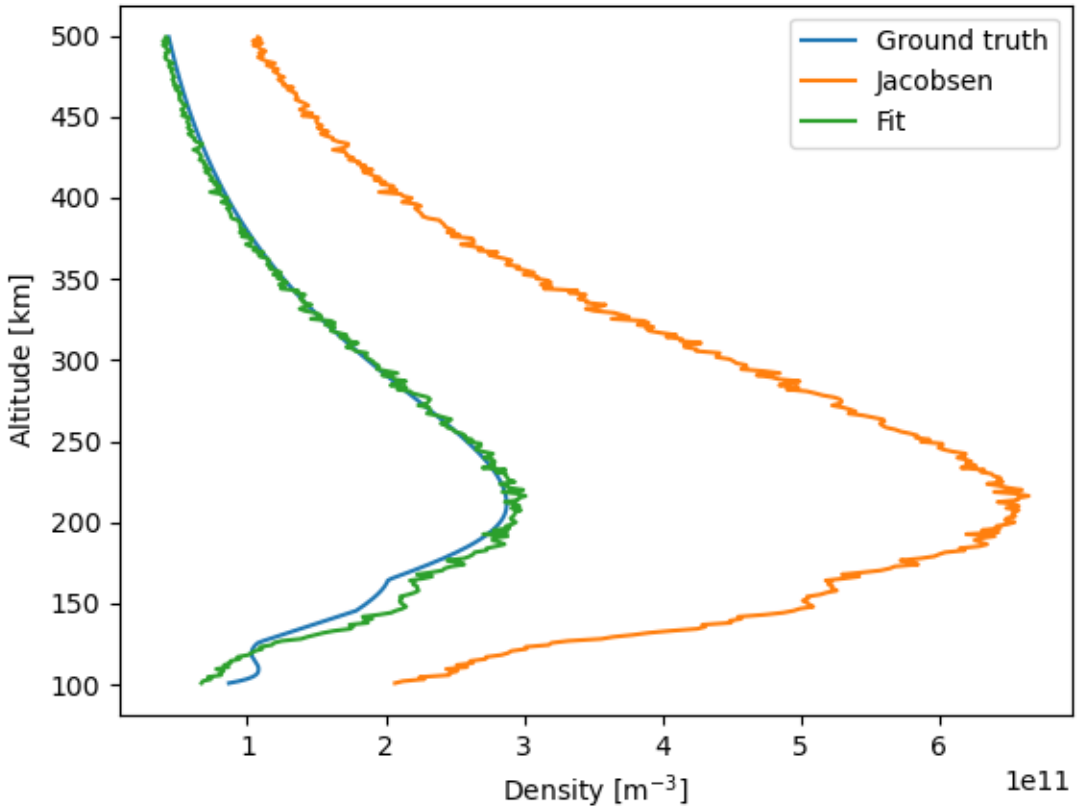


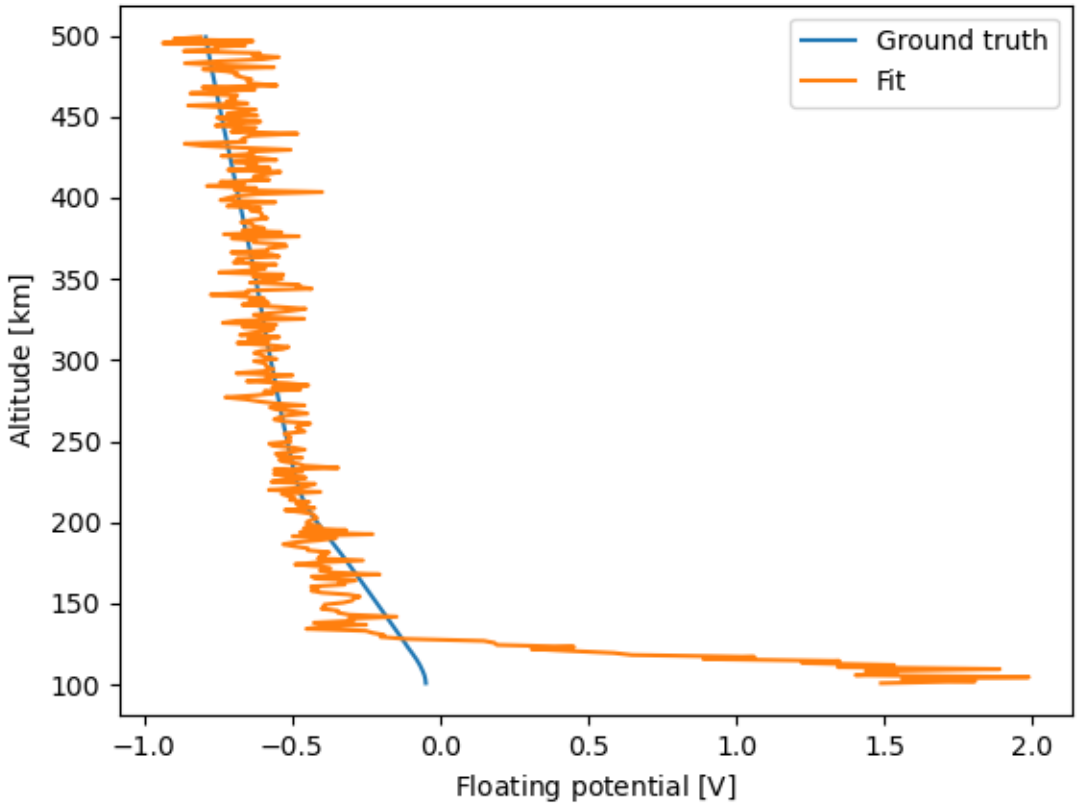
The density agrees well with both the ground truth, as well as densities inferred with the Jacobsen-Bekkeng method. Close inspection, however, reveals a small discrepancy between our method and the Jacobsen-Bekkeng method. This is due to the fact that the system is overdetermined (inferring 2 parameters from 4 measurements), and the Jacobsen-Bekkeng method minimizes a squared residual in $\mathrm{d}I^2/\mathrm{d}V$, whereas our method minimizes a squared residual in I itself. Removing two of the bias voltages lead to perfect agreement. The method captures a trend in the floating potential but cannot make accurate predictions of it.

Now that the technique is established, we can proceed by assuming that the finite-length model is a perfect representation of reality, and fitting the currents to the finite-length characteristic. This is simply a matter of substituting the following lines:

```
model_truth = finite_length_current
model_pred = finite_length_current
```

During data synthesis we will receive warnings about the normalized voltage eV/kT exceeding the maximum of 100 in the finite-length model. This happens for the lower altitudes, when the temperature T is low. It will not prevent execution, however, but it is important to be aware of, since it means the model must extrapolate, which is less accurate than interpolation. The resulting inference is plotted as before:


Ground truth
Jacobsen
Fit
Altitude [km]
500
450
400
350
300
250
200
150
100
1
2
3
4
5
6
1e11
Density [m^{-3}]



As is to be expected, the inferred density is close to the ground truth. For finite-length effects, the inferred density is not entirely independent of the specified temperature `T`, and this causes some error. The dependence is weak enough, however, that the error is not severe. The accuracy is also degraded for lower altitudes due to the aforementioned extrapolation. The floating potential is not very accurate, but then again, this cannot be expected when it was not accurate for the simpler case. Finally, it is interesting to compare with the Jacobsen-Bekkeng method, since this is indicative of the error caused by neglecting end effects.

6.4.4 Inversion by machine learning

Another way to solve the inverse problem is by machine learning, or more specifically, by regression. This was first described in [Chalaturnyk] and [Guthrie], and we shall carry out a similar (though not entirely identical) procedure here. We consider a plasma parameter (here: the electron density n_e) to be approximated by some function f of measured currents:

$$n_e = f(\hat{I}_1, \hat{I}_2, ..., \hat{I}_N)$$

The function represents, in our case, a machine learning network, and has a number of coefficients that will be determined by fitting (training) it to synthetic data with known densities n_e. Once the network is trained, it can be used to predict densities $\hat{n}_e$ from actual measurements. The procedure can be split in three parts, as seen in the example code:

```
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
import langmuir as l
```

(continues on next page)

(continued from previous page)

```
from tqdm import tqdm
from itertools import count
from localreg import RBFnet, plot_corr
from localreg.metrics import rms_rel_error

geo = l.Cylinder(r=0.255e-3, l=25e-3, lguard=float('inf'))
model = l.finite_length_current
Vs = np.array([2, 3, 4, 5]) # bias voltages

"""
PART 1: GENERATE SYNTHETIC DATA USING LANGMUIR
"""

def rand_uniform(N, range):
    """Generate N uniformly distributed numbers in range"""
    return range[0]+(range[1]-range[0])*np.random.rand(N)

def rand_log(N, range):
    """Generate N logarithmically distributed numbers in range"""
    x = rand_uniform(N, np.log(range))
    return np.exp(x)

N = 1000
ns  = rand_log(N, [1e11, 12e11])  # densities
Ts = rand_uniform(N, [800, 2500]) # temperatures
V0s = rand_uniform(N, [-3,  0])   # floating potentials

# Generate probe currents corresponding to plasma parameters
Is = np.zeros((N,len(Vs)))
for i, n, T, V0 in zip(count(), ns, Ts, tqdm(V0s)):
    Is[i] = model(geo, l.Electron(n=n, T=T), V=V0+Vs)

"""
PART 2: TRAIN AND TEST THE REGRESSION NETWORK
"""

# Use M first data points for training, the rest for testing.
M = int(0.8*N)

# Train by minimizing relative error in density
net = RBFnet()
net.train(Is[:M], ns[:M], num=20, relative=True, measure=rms_rel_error)

# Plot and print error metrics on test data
pred = net.predict(Is[M:])

fig, ax = plt.subplots()
plot_corr(ax, ns[M:], pred, log=True)
print("RMS of relative density error: {:.1f}%".format(100*rms_rel_error(ns[M:],␣
↪pred)))

"""
PART 3: PREDICT PLASMA PARAMETERS FROM ACTUAL DATA
"""

data = l.generate_synthetic_data(geo, Vs, model=model)
pred = net.predict(data['I'])
```

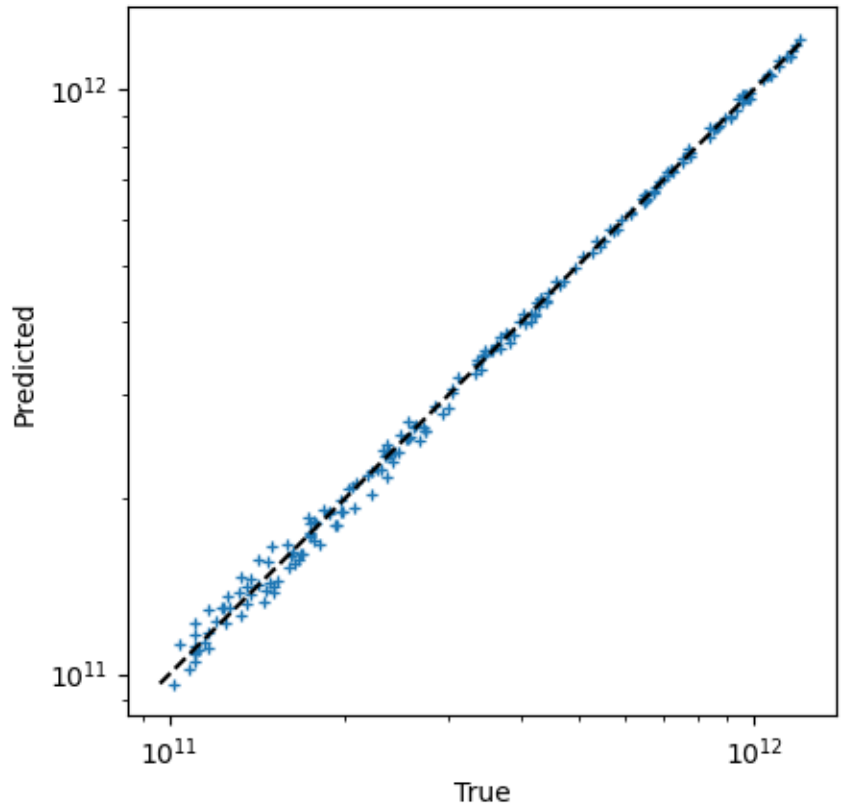
(continues on next page)

(continued from previous page)

```
plt.figure()
plt.plot(data['ne'], data['alt'], label='Ground truth')
plt.plot(pred, data['alt'], label='Predicted')
plt.xlabel('Density $[\mathrm{m}^{-3}]$')
plt.ylabel('Altitude $[\mathrm{km}]$')
plt.legend()

plt.show()
```

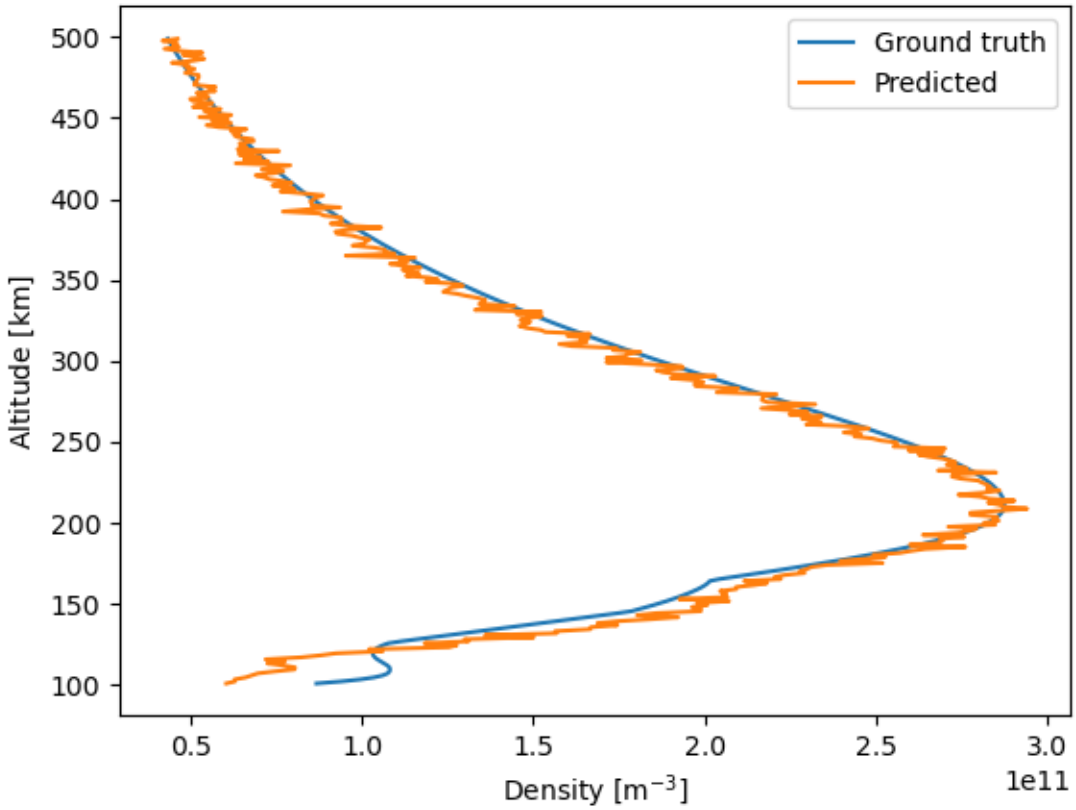
First, N synthetic data points are generated by randomly selecting densities (n s), temperatures (T s) and floating potentials (V_0 s) from appropriate ranges, and computing corresponding currents using Langmuir. Beware that if the ranges do not properly cover the values expected in the actual data, the network may have to extrapolate, which usually result in inaccurate predictions.

Second, we train a regression network. We use a *radial basis function* (RBF) network from the localreg library for closer proximity to the above-mentioned research, although one could also use TensorFlow or other alternatives. Remember, however, that from a machine learning point-of-view, this is a small problem that is not well served by *deep* neural networks with many degrees of freedom. Regardless of choice, it is important to test the network's performance (quantitatively!) on data which was not used in training. We use 80% of the synthetic data for training, and set aside the remaining part for testing. The testing reveal that the root mean square (RMS) of the relative density error is about 3–4% (depending on the seed of the random number generator). The correlation plot also show good agreement between prediction and ground truth:



Third and finally, once the network has been tested, it can be used to make predictions on real data (or in our case data

generated with `generate_synthetic_data()`):



For real applications it is advisable to do the last step in a separate file, using a pre-trained network that is stored as a file for reuse.

When a parameter (e.g., the density) may span multiple orders of magnitudes, often only the most significant figures are of interest. $1.00 \cdot 10^{12}\,\mathrm{m}^{-3}$ and $0.99 \cdot 10^{12}\,\mathrm{m}^{-3}$ are in practice indistinguishable, whereas there's a huge difference between $1 \cdot 10^{10}\,\mathrm{m}^{-3}$ and $2 \cdot 10^{10}\,\mathrm{m}^{-3}$, although the difference between the two pairs of numbers are the same. In such cases it makes sense to minimize the *relative error*

$$\frac{\hat{n}_e - n_e}{n_e},$$

rather than the absolute error $\hat{n}_e - n_e$. Although the range of densities in our example is not that large, we choose to train the network using relative errors. We also distribute the training densities logarithmically, since otherwise, few data points would have low orders of magnitude, which would lead to an under-prioritization of the accuracy of of these lower density magnitudes.

CHAPTER 7

Citing Langmuir

If you use Langmuir for your research, please cite the version of Langmuir you used to produce your results. Langmuir is permanently archived with its own *Digital Object Identifier* (DOI) at Zenodo.

Furthermore, you should cite the works behind each model you use:

- For OML theory cite [MottSmith] for Maxwellian plasmas, or if you use Kappa-Cairns distributed plasmas, [Darian].
- For finite-radius models cite [Laframboise] for Maxwellian plasmas, or if you use Kappa-Cairns distributed plasmas, [Darian].
- For finite-length models cite [Marholm].

CHAPTER 8

Bibliography

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Bibliography

[Marholm] S. Marholm and R. Marchand, *Finite-length effects on cylindrical Langmuir probes*, Physical Review Research, 2, 2020.

[Marholm2] S. Marholm, *The Unstructured Particle-In-Cell Method with Applications to Objects in Ionospheric Plasmas*, PhD dissertation, University of Oslo, 2020.

[Darian] D. Darian, S. Marholm, M. Mortensen and W.J. Miloch, *Theory and simulations of spherical and cylindrical Langmuir probes in non-Maxwellian plasmas*, Plasma Physics and Controlled Fusion, 61, 8, 2019.

[Laframboise] J. Laframboise, *Theory of spherical and cylindrical Langmuir probes in a collisionless maxwellian plasma at rest*, PhD dissertation, University of Toronto, 1966.

[MottSmith] H.M. Mott-Smith, I. Langmuir, *The theory of collectors in gaseous discharges*, Physical Review, 28, 1926.

[Whipple] E.C. Whipple, *Potentials of surfaces in space*, Reports on Progress in Physics, 44, 1981.

[Bekkeng] T.A. Bekkeng, *Development of a miniaturized multi-Needle Langmuir Probe system for in-situ measurements of electron density and spacecraft floating potential*, PhD dissertation, University of Oslo, 2017.

[Bittencourt] J.A. Bittencourt, *Funamentals of Plasma Physics*, Third edition, Springer, 2004.

[Ikezi] Ikezi et al., *Probe Noise in Quiescent Plasmas*, Journal of the Physical Society of Japan, 25, 6, 1968.

[Jacobsen] K.S.Jacobsen et al., *A new Langmuir probe concept for rapid sampling of space plasma electron density*, Measurement Science and Technology, 21, 2010.

[Hoang] H. Hoang et al., *A study of data analysis techniques for the multi-needle Langmuir probe*, Measurement Science and Technology, 29, 2019.

[Barjatya] A. Barjatya et al., *Invited article: Data analysis of the floating potential measurement unit abourd the international space station*, Review of Scientific Instruments, 80, 4, 2009.

[IRI] The IRI Ionospheric Model.

[MSISE] The MSISE Ionospheric Model.

[Chalaturnyk] J. Chalaturnyk and R. Marchand, *A First Assessment of a Regression-Based Interpretation of Langmuir Probe Measurements*, Frontiers in Physics, 7, 2019.

[Guthrie] J. Guthrie, R. Marchand and S. Marholm, *Inference of plasma parameters from fixed-bias multi-needle Langmuir probes (m-NLP)*, Measurement Science and Technology, 32, 2021.