Click here to view the PDF version.

```julia
# Block installing Julia packages needed to run this notebook.
begin
    using Plots
    using PlutoUI
    using PlutoTeachingTools
    using LaTeXStrings
    using Symbolics
    using NLsolve
    using HypertextLiteral: @htl, @htl_str
end
```

## Table of Contents

**Introduction**
- Opening remarks
- Interpolating data
- Taking derivatives
- Solving differential equations
- Modelling electric circuits

# Introduction

## Opening remarks

Slides of the presentation: These summarise the following paragraphs and the content of the course in a short presentation. TODO: Add slide link here.

In modern practice **every scientist employs computers**. Some use them to symbolically derive equations for complex theories, others perform simulations, yet others analyse and visualise experimental results. In all these cases we encounter **mathematical problems**, which we need to **solve using numerical techniques**. The point of this lecture is to learn how to **formalise such numerical procedures** mathematically, **implement them** using the Julia programming language and **analyse using pen and paper** why some methods work better, some worse and some not at all.

You might think learning about this is a bit of an overkill and probably not very useful for your future studies. But let us look at a few examples, which give an overview what we will study in more depth throughout the course.

## Interpolating data

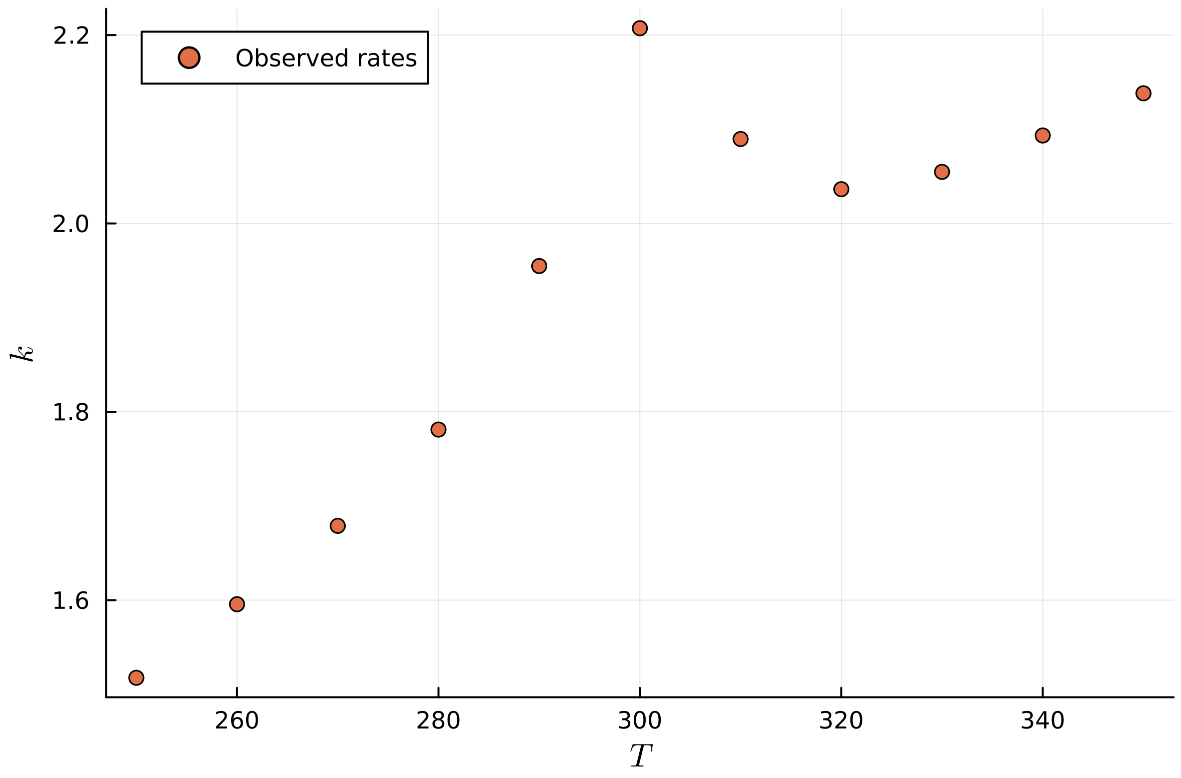
We start with the following problem: Imagine we did some measurement of the rate of a chemical reation at different temperatures, let's say

```julia
data = """
# Temperature(K)  Rate(1/s)
  250.0           1.51756
  260.0           1.59566
  270.0           1.67888
  280.0           1.78110
  290.0           1.95476
  300.0           2.20728
  310.0           2.08967
  320.0           2.03635
  330.0           2.05474
  340.0           2.09338
  350.0           2.13819
""";
```

Now we are curious about the rates at another temperature, say $255K$. The basic laws of thermodynamics and chemical kinetics tell us that the rate should be a *continuous function* of the temperature. For example Arrhenius' law

$$k(T) = A \exp\left(-\frac{E_A}{k_B T}\right) \tag{4}$$

establishes such a relationship, where $A$, $E_A$ and $k_B$ are some constants. Note, that in this case the behaviour is more complicated as is immediately obvious from a plot of the observed data:



Still, it seems reasonable that we should be able to **determine some continuous function $f$ from the observed data**, which **establishes a good model** for the relationship $k = f(T)$. Albeit $255K$ has not been measured, we can thus evaluate $f(255K)$ and get an estimate for the rate at this temperature. Since the new temperature $255K$ islocated *within* the observed data range (i.e. here $250K$ to $350K$) we call this procedure **interpolation**.

In this lecture we will discuss some common interpolation techniques. In particular we will develop the **mathematical formalism** for these techniques and — most importantly — we will use this formalism to understand conditions when these methods work and **when these methods fail**.

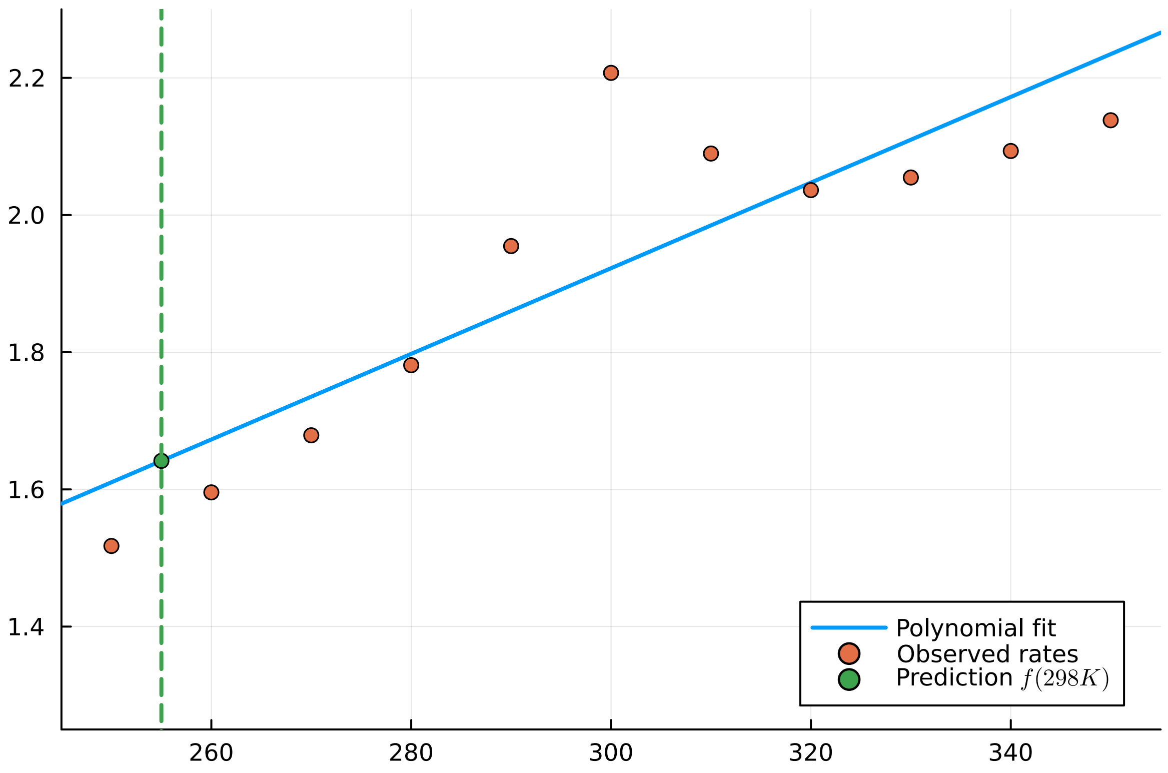
In fact our example here is already a little tricky. Let us illustrate this point for **polynomial interpolation**, one particularly frequent technique. Without going into details for now, one basic idea is to fit a polynomial of degree $N$

$$k(T) = \sum_{n=1}^{N} \alpha_n T^n$$

to the data. By some procedure discussed later we thus determine the unknown polynomial coefficients $\alpha_n$, such that the polynomial best matches our observations and use that to determine $f(255K)$.

Let's see the result of such a fit. The slider let's you play with the polynomial order:

- `N =` 1



Clearly, the quality of such a fit depends strongly on the polynomial order. Perhaps suprising is, however, that **higher degree is not necessarily better**. We will discuss why this is.

**Related questions we will discuss:**

- How can I fit a polynomial to data points ?

- How accurate can I expect such a polynomial fit to be ?
- Can I choose an optimal polynomial order to obtain the most accurate answer ?
- If I have control over the strategy to acquire data (e.g. how to design my lab experiment), can I have an influence on the accuracy of such interpolations ?

## Taking derivatives

Derivatices characterise the change of a quantity of interest. E.g. the acceleration as a derivative of the velocity indeed characterises how the velocity changes over time.

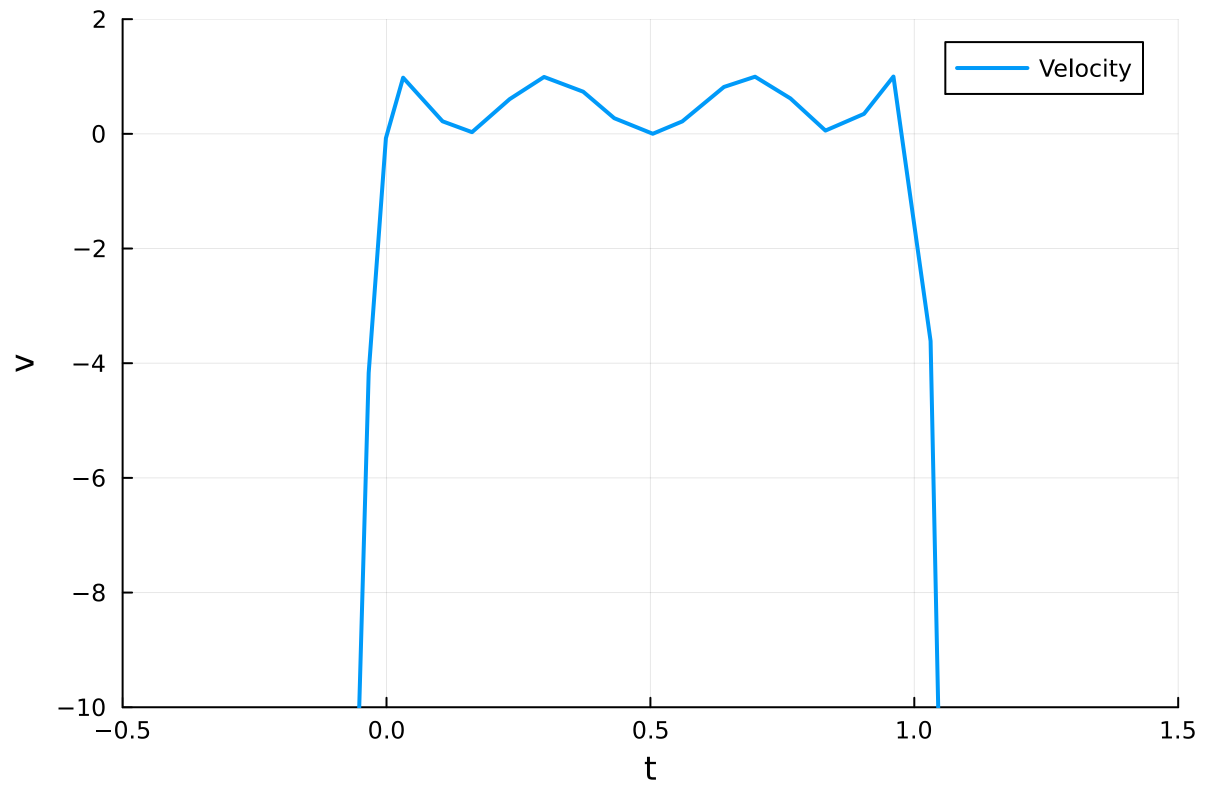
In a nutshel understanding changes of a materials or any other physical system as we interact with it, is the essential goal of pretty much every field of science and engineering.

As a result derivatives are everywhere. But computing derivatives by hand is not always easy. To see this consider the **innocent-looking velocity function**.

```
v (generic function with 1 method)
```

```
v(t) = 64t * (1 - t) * (1 - 2t)^2 * (1 - 8t + 8t^2)^2
```

We can plot it to get an idea:



```
let
    p = plot(v;
             ylims=(-10, 2),      # Limit on y axis
             xlims=(-0.5, 1.5),   # Limit on x axis
             linewidth=2,         # Width of the blue line
             label="Velocity",    # Label of the graph
             xlabel="t",
             ylabel="v")
end
```

Clearly between $0$ and $1$ the function changes quite rapidly. To investigate the **acceleration** there **we take the derivative**.

This can be done by hand, but instead we will employ a Julia package (namely `Symbolics`) to take the derivative for us. The result is:

$$64\left(1-8t+8t^2\right)^2(1-2t)^2\,(1-t)-64\left(1-8t+8t^2\right)^2(1-2t)^2t-256\left(1-8t+8t^2\right)^2t\,(1$$

```
let
    @variables t          # Define a variable
    dt = Differential(t)  # Differentiating wrt. t
    dv_dt = dt( v(t) )    # Compute dv / dt
    expand_derivatives(dv_dt)
end
```

Clearly far from a handy expression and not the kind of derivative one wants to compute by hand.

So let us **compute this derivative numerically** instead. An idea to do so goes back to the definition of the derivative $v'(t)$ as the limit $h \to 0$ of the slope of secants over an intervall $[t, t+h]$, i.e.

$$v'(t) = \lim_{h\to 0} \frac{v(t+h) - v(t)}{h}.$$

A natural idea is to not fully take the limit, i.e. to take a small $h > 0$ and approximate

$$v'(x) \approx \frac{v(t+h) - v(t)}{h}.$$

The expectation is that as we take smaller and **smaller values for $h$**, this **converges to the exact derivatives**.

So let's try this at the point `t0 = 0.2` for various values for `h`

```
h_values =
▸[1.0e-15, 3.16228e-15, 1.0e-14, 3.16228e-14, 1.0e-13, 3.16228e-13, 1.0e-12, 3.16228e-12,
```

```julia
h_values = 10 .^ (-15:0.5:1)
```

We compute the derivatives at each value for $h$ ...

```
derivatives =
▸[8.99281, 9.0755, 9.05942, 9.06497, 9.06719, 9.06567, 9.06603, 9.06608, 9.06609, 9.06609,
```

```julia
derivatives = [ ( v(t0 + h) - v(t0) )/h for h in h_values ]
```

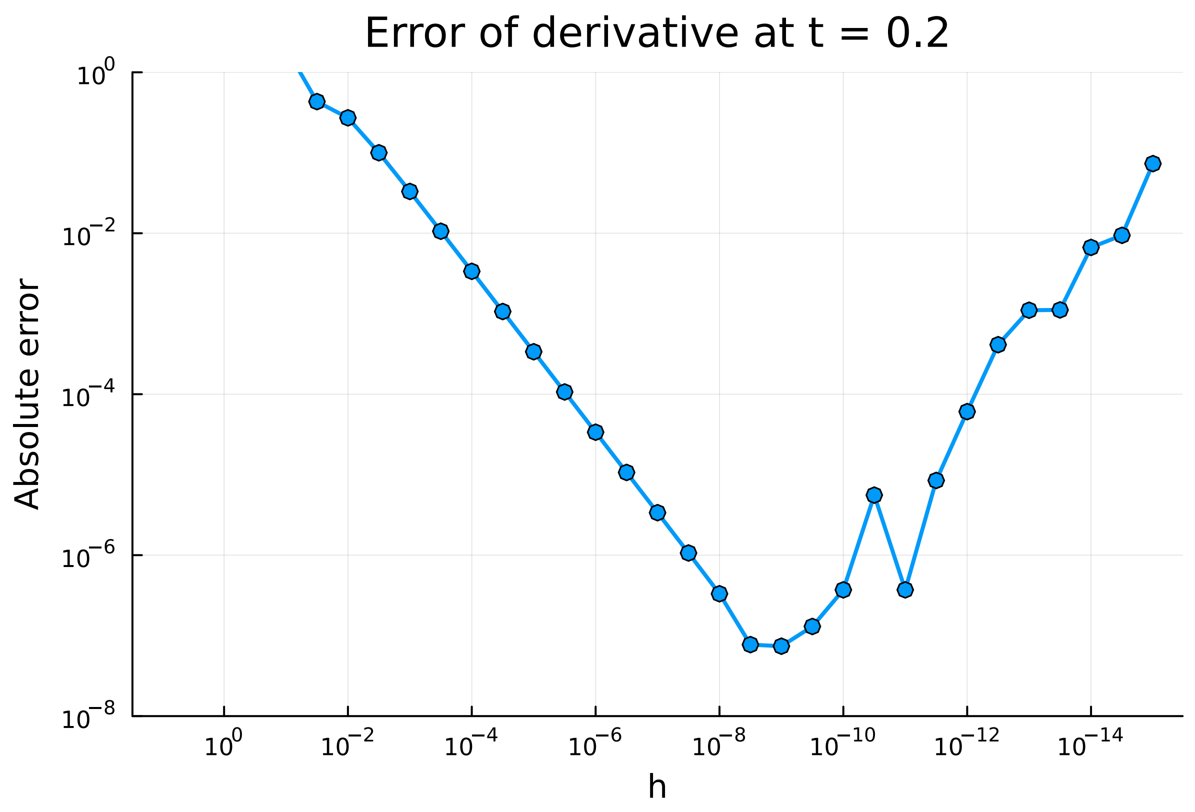
... and compute the error against a reference value:

```
▸[0.0732799, 0.00941812, 0.00666652, 0.00111438, 0.00110504, 0.000412217, 6.06921e-5, 8.47
```

```julia
begin
    using ForwardDiff # Package for computing derivatives of Julia functions
    reference = ForwardDiff.derivative(v, t0)

    errors = [abs(d - reference) for d in derivatives]
end
```

Finally we plot the errors on a log-log plot:



```julia
let
    plot(h_values, errors;
        xaxis=:log, yaxis=:log,
        xflip=true,  # Flip order of x axis
        ylims=(1e-8, 1),
        mark=:o,     # Mark all points by circle
        xlabel="h",
        ylabel="Absolute error",
        linewidth=2,
        label="",
        xticks=10.0 .^ (-16:2:0),
        title="Error of derivative at t = $t0",
    )
end
```

We observe that while indeed initially the error decreases as $h$ decreases at some point it starts to increase again with **results worse and worse**.

With this slider you can check this is indeed the case for pretty much all values of $t$ where we take the derivative numerically:

t0 = 0.2

## Related questions we will discuss:

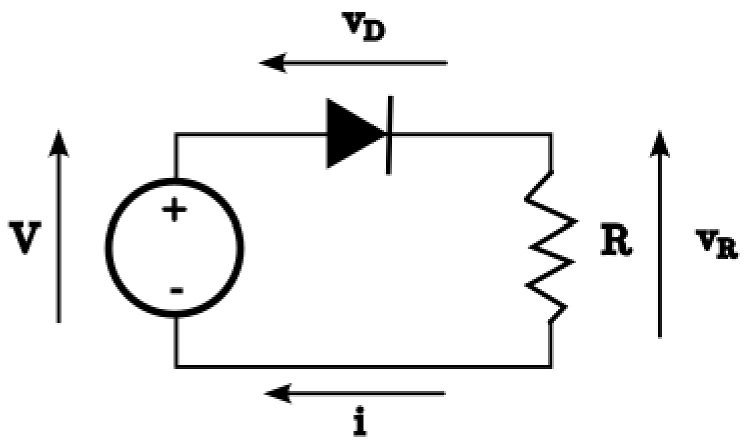
- There seems to be some optimal value for $h$. Is it independent of the function to be differentiated ?
- Here seems to be some minimal error we can achieve. Are there more accurate derivative formulas ?
- The convergence plot as $h$ decreases seems to have the same slope (convergence rate) for all points $t$. Does this slope depend on $f$ ? How can we reach faster convergence ?

## Solving differential equations

TODO Some motivating ODE example

## Modelling electric circuits

Let us consider the simple Diode model circuit



which consists of

- Some voltage source, generating a voltage $V$
- A resistor $R$ with the linear relationship

$$v_R = Ri \tag{1}$$

  between the current $i$ and its voltage $v_R$.
- A diode for which we will take the standard Shockley diode model, i.e. the equation

$$i = i_0(e^{v_D/v_0} - 1) \tag{2}$$

between the diode's current $i$ and voltage $v_D$. $v_0$ and $i_0$ are parameters, which characterise the diode.

We now want to solve this problem numerically. This notebook already defines variables for the parameters $R$, $V$, $i_0$ and $v_0$. This is done using sliders, which you find further down this notebook. Currently these variables have the values

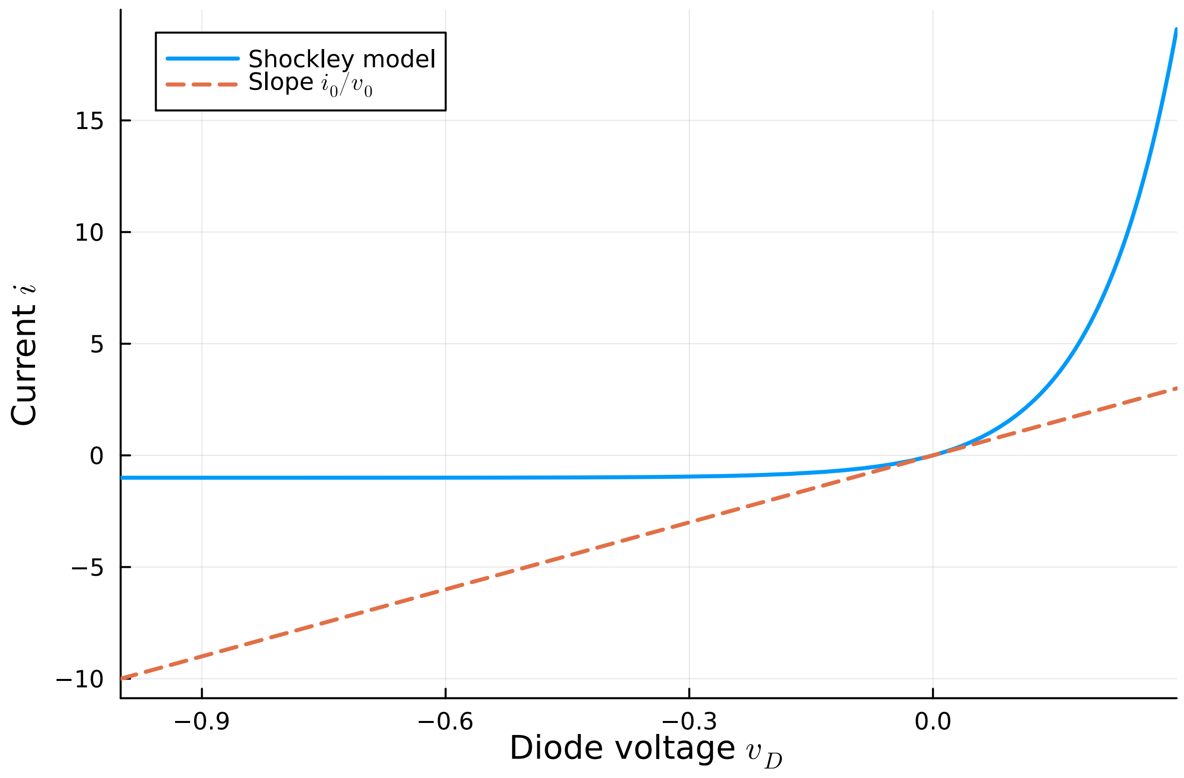
- `R` = 1.0
- `V` = 1.0
- `i0` = 1.0
- `v0` = 0.1

In Julia code we can thus define the Shockley diode relationship, mapping $v_D$ to $i$ as follows

```
shockley_diode (generic function with 1 method)
```

```julia
function shockley_diode(vD)
    i0 * (exp(vD/v0) - 1)
end
```

Graphically for above parameters this looks like



```julia
let
    p = plot(shockley_diode, xlim=(-1, 0.3);
             label="Shockley model",
             xlabel=L"Diode voltage $v_D$",
             ylabel=L"Current $i$",
             linewidth=2)
    plot!(p, v -> (i0/v0) * v;
          label=L"Slope $i_0 / v_0$",
          linestyle=:dash,
          linewidth=2)
end
```

Our goal is to model this circuit, i.e. understand its current $i$ the voltages across the circuit elements.

First we balance the supplied voltage $V$ across the circuit. Using equations (1) and (2) this leads to the following:

$$V = v_R + v_D = R\,i + v_D = R\,i_0\left(e^{v_D/v_0} - 1\right) + v_D \tag{3}$$

We introduce the function

$$f(v_D) = R\,i_0\left(e^{v_D/v_0} - 1\right) + v_D - V,$$

which makes (3) equivalent to seeking a diode voltage $v_D$ such that $f(v_D) = 0$ — a **root finding problem**.

However, since $f$ is not just a simple polynomial, but a **non-linear function** there is no explicit analytic formula for finding its solution. We are thus forced to employ numerical methods.

As we will see in the next lectures there are a number of numerical techniques to solve such nonlinear problems $f(x) = 0$. However, similar to polynomial fitting, none of them always just work. In this case we are lucky and can actually directly defer to a standard package of the Julia programming language, called `NLsolve` to solve our problem:

```
▸ [0.0659611]
```

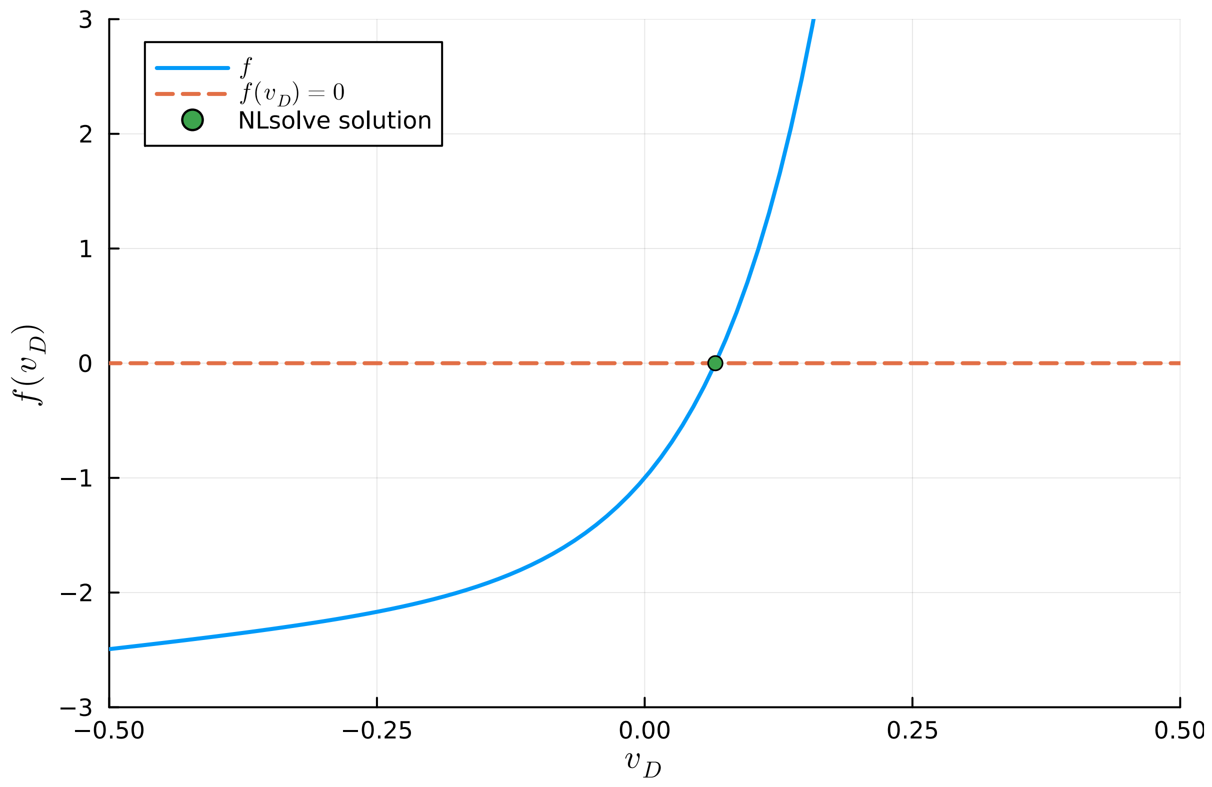
```julia
begin
    # Define our function
    f(vD) = R*i0*(exp(vD/v0) - 1) + vD - V

    # Solve it with nlsolve
    z = nlsolve(v -> f(v[1]), [0.0]).zero
end
```

which for the parameters

- `i0 =` 1.0
- `v0 =` 0.1
- `R =` 1.0
- `V =` 1.0

tells us the root is located at $v_D \simeq$ 0.066. We can also check this graphically:



```julia
let
    vs = range(-0.5, 0.5; length=100)
    plot(vs, f.(vs); xlims=(-0.5, 0.5), ylims=(-3, 3), legend=:topleft,
              label=L"f", linewidth=2, xlabel=L"v_D", ylabel=L"f(v_D)")
    hline!([0], linestyle=:dash, linewidth=2, label=L"f(v_D) = 0")
    scatter!(z, [0], label="NLsolve solution")
end
```

Given this nice `NLsolve` package, which seems to do the trick automatically, you might wonder **why we even bother** discussing, analysing and implementing such techniques at all.

Well, unfortunately **for many realistic numerical settings** (e.g. the quantum-chemical simulation of materials) even **standard packages** such as NLsolve are **no longer able to automatically figure out the best solution strategy**. As a result developing some intuition for the required numerical procedure.

Even if you now might think, that your heart beats for **experimental research** and you will never need numerics, keep in mind that **all data analysis** uses procedures **based on such techniques**. **Even commercial packages** for experimental post-processing rely heavily on the procedures we will cover and **sometimes get it wrong}}. See for example this report on Microsoft Excel). This** **can and has compromised scientific fundings in the past**. Therefore it's **best to be prepared**, so you can judge what is wrong: The experiment or the numerics.

**Related questions we will discuss:**

- How can I understand whether an obtained numerical answer is credible ?
- What techniques based on visualisation or plotting help me to understand the accuracy of a numerical algorithm ?
- How can I overcome numerical issues ?
- What are numerically stable techniques for interpolation, basic data analysis or solving standard scientific problems ?
- How can I understand and evaluate the speed of convergence of an algorithm and improve it even further ?

**Numerical analysis**

1. Introduction
2. The Julia programming language
3. Revision and preliminaries
4. Root finding and fixed-point problems
5. Interpolation
6. Direct methods for linear systems
7. Iterative methods for linear systems
8. Eigenvalue problems
9. Numerical integration
10. Numerical differentiation
11. Initial value problems
12. Boundary value problems