homework 9, version 1

Submission by: ***Neham Soni*** (neham@mit.edu)

Homework 9: *Climate modeling I*

`18.S191`, fall 2020

```
student = ▸(name = "Neham Soni", kerberos_id = "neham")
```

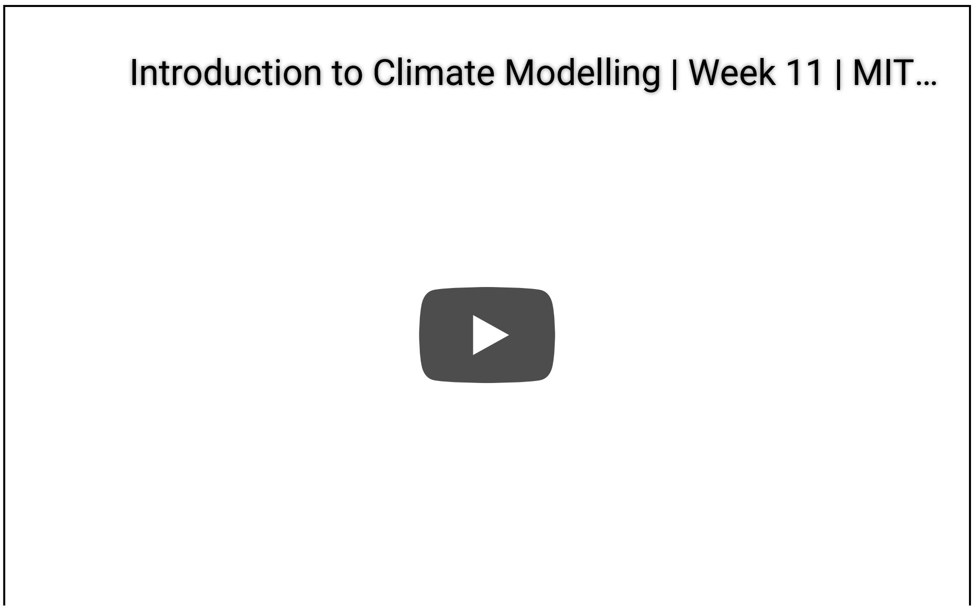
```
# edit the code below to set your name and kerberos ID (i.e. email without @mit.edu)

student = (name = "Neham Soni", kerberos_id = "neham")

# you might need to wait until all other cells in this notebook have completed running.
# scroll around the page to see what's up
```

Let's create a package environment:

```
begin
    import Pkg
    Pkg.activate(mktempdir())
    Pkg.add([
            "Plots",
            "PlutoUI",
            "LaTeXStrings",
            "Distributions",
            "Random",
    ])
    using LaTeXStrings
    using Plots
    using PlutoUI
    using Random, Distributions
end
```



Before working on the homework, make sure that you have watched the first lecture on climate modeling 👆 . We have included the important functions from this lecture notebook in the next cell. Feel free to have a look!

```
Main.workspace1359.Model
```

```julia
module Model

const S = 1368; # solar insolation [W/m^2]  (energy per unit time per unit area)
const α = 0.3; # albedo, or planetary reflectivity [unitless]
const B = -1.3; # climate feedback parameter [W/m^2/°C],
const T0 = 14.; # preindustrial temperature [°C]

absorbed_solar_radiation(; α=α, S=S) = S*(1 - α)/4; # [W/m^2]
outgoing_thermal_radiation(T; A=A, B=B) = A - B*T;

const A = S*(1. - α)/4 + B*T0; # [W/m^2].

greenhouse_effect(CO2; a=a, CO2_PI=CO2_PI) = a*log(CO2/CO2_PI);

const a = 5.0; # CO2 forcing coefficient [W/m^2]
const CO2_PI = 280.; # preindustrial CO2 concentration [parts per million; ppm];
CO2_const(t) = CO2_PI; # constant CO2 concentrations

const C = 51.; # atmosphere and upper-ocean heat capacity [J/m^2/°C]

function timestep!(ebm)
    append!(ebm.T, ebm.T[end] + ebm.Δt*tendency(ebm));
    append!(ebm.t, ebm.t[end] + ebm.Δt);
end;

tendency(ebm) = (1. /ebm.C) * (
    + absorbed_solar_radiation(α=ebm.α, S=ebm.S)
    - outgoing_thermal_radiation(ebm.T[end], A=ebm.A, B=ebm.B)
    + greenhouse_effect(ebm.CO2(ebm.t[end]), a=ebm.a, CO2_PI=ebm.CO2_PI)
);

begin
    mutable struct EBM
        T::Array{Float64, 1}

        t::Array{Float64, 1}
        Δt::Float64

        CO2::Function

        C::Float64
        a::Float64
        A::Float64
        B::Float64
        CO2_PI::Float64

        α::Float64
        S::Float64
    end;

    # Make constant parameters optional kwargs
    EBM(T::Array{Float64, 1}, t::Array{Float64, 1}, Δt::Real, CO2::Function;
        C=C, a=a, A=A, B=B, CO2_PI=CO2_PI, α=α, S=S) = (
        EBM(T, t, Δt, CO2, C, a, A, B, CO2_PI, α, S)
    );

    # Construct from float inputs for convenience
    EBM(T0::Real, t0::Real, Δt::Real, CO2::Function;
        C=C, a=a, A=A, B=B, CO2_PI=CO2_PI, α=α, S=S) = (
        EBM(Float64[T0], Float64[t0], Δt, CO2;
            C=C, a=a, A=A, B=B, CO2_PI=CO2_PI, α=α, S=S);
    );
end;
```

```
begin
    function run!(ebm::EBM, end_year::Real)
        while ebm.t[end] < end_year
            timestep!(ebm)
        end
    end;

    run!(ebm) = run!(ebm, 200.) # run for 200 years by default
end




CO2_hist(t) = CO2_PI * (1 .+ fractional_increase(t));
fractional_increase(t) = ((t .- 1850.)/220).^3;

begin
    CO2_RCP26(t) = CO2_PI * (1 .+ fractional_increase(t) .* min.(1., exp.(-((t
.-1850.).-170)/100))) ;
    RCP26 = EBM(T0, 1850., 1., CO2_RCP26)
    run!(RCP26, 2100.)

    CO2_RCP85(t) = CO2_PI * (1 .+ fractional_increase(t) .* max.(1., exp.(((t
.-1850.).-170)/100)));
    RCP85 = EBM(T0, 1850., 1., CO2_RCP85)
    run!(RCP85, 2100.)
end

end
```

Exercise 1 - *policy goals under uncertainty*

A recent ground-breaking **review paper** produced the most comprehensive and up-to-date estimate of the *climate feedback parameter*, which they find to be

$$B \approx \mathcal{N}(-1.3, 0.4),$$

i.e. our knowledge of the real value is normally distributed with a mean value $\overline{B} = -1.3$ W/m²/K and a standard deviation $\sigma = 0.4$ W/m²/K. These values are not very intuitive, so let us convert them into more policy-relevant numbers.

Definition: *Equilibrium climate sensitivity (ECS)* is defined as the amount of warming ΔT caused by a doubling of CO_2 (e.g. from the pre-industrial value 280 ppm to 560 ppm), at equilibrium.

At equilibrium, the energy balance model equation is:

$$0 = \frac{S(1-\alpha)}{4} - (A - BT_{eq}) + a \ln\left(\frac{2\ \text{CO}_{2\text{PI}}}{\text{CO}_{2\text{PI}}}\right)$$

From this, we subtract the preindustrial energy balance, which is given by:

$$0 = \frac{S(1-\alpha)}{4} - (A - BT_0),$$

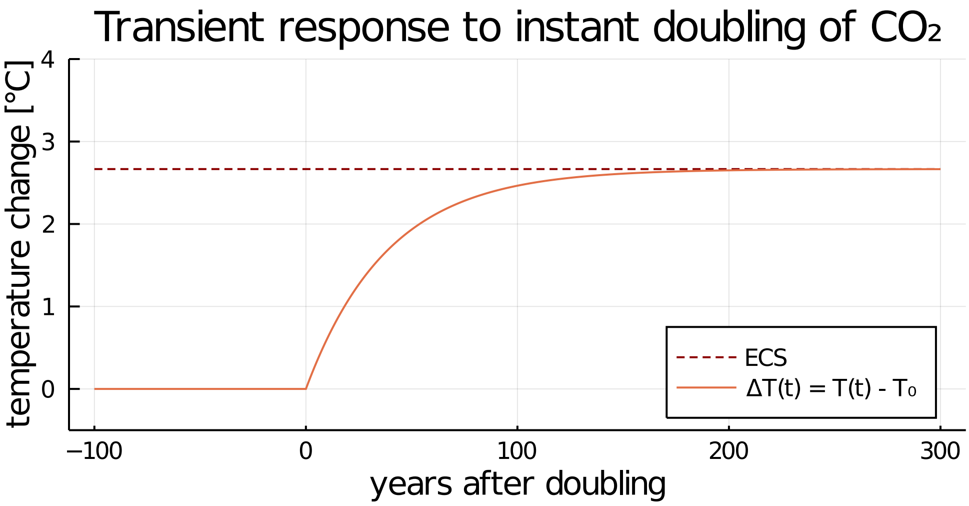
The result of this subtraction, after rearranging, is our definition of ECS:

$$\text{ECS} \equiv T_{eq} - T_0 = -\frac{a \ln(2)}{B}$$

```
ECS(; B=B̄, a=Model.a) = -a*log(2.)./B;
```

The plot below provides an example of an "abrupt 2xCO_2" experiment, a classic experimental treatment method in climate modelling which is used in practice to estimate ECS for a particular model. (Note: in complicated climate models the values of the parameters a and B are not specified *a priori*, but *emerge* as outputs of the simulation.)

The simulation begins at the preindustrial equilibrium, i.e. a temperature $T_0 = 14$°C is in balance with the pre-industrial CO_2 concentration of 280 ppm until CO_2 is abruptly doubled from 280 ppm to 560 ppm. The climate responds by warming rapidly, and after a few hundred years approaches the equilibrium climate sensitivity value, by definition.



$B =$ -1.3

Exercise 1.1 - *Develop understanding for feedbacks and climate sensitivity*

👉 Change the value of B using the slider above. What does it mean for a climate system to have a more negative value of B? Explain why we call B the *climate feedback parameter*.

`observations_from_changing_B =`

B the climate feedback parameter can be thouth of a self correcting parameter in a model, i.e rate of cange of tem depends upon the feedback, while the opposite sign implies its tendency to return at a stable point. Smaller the Value that is larger the negitive slope making it more easier and faster to stabelize.

👉 What happens when B is greater than or equal to zero?

`observations_from_nonnegative_B =`

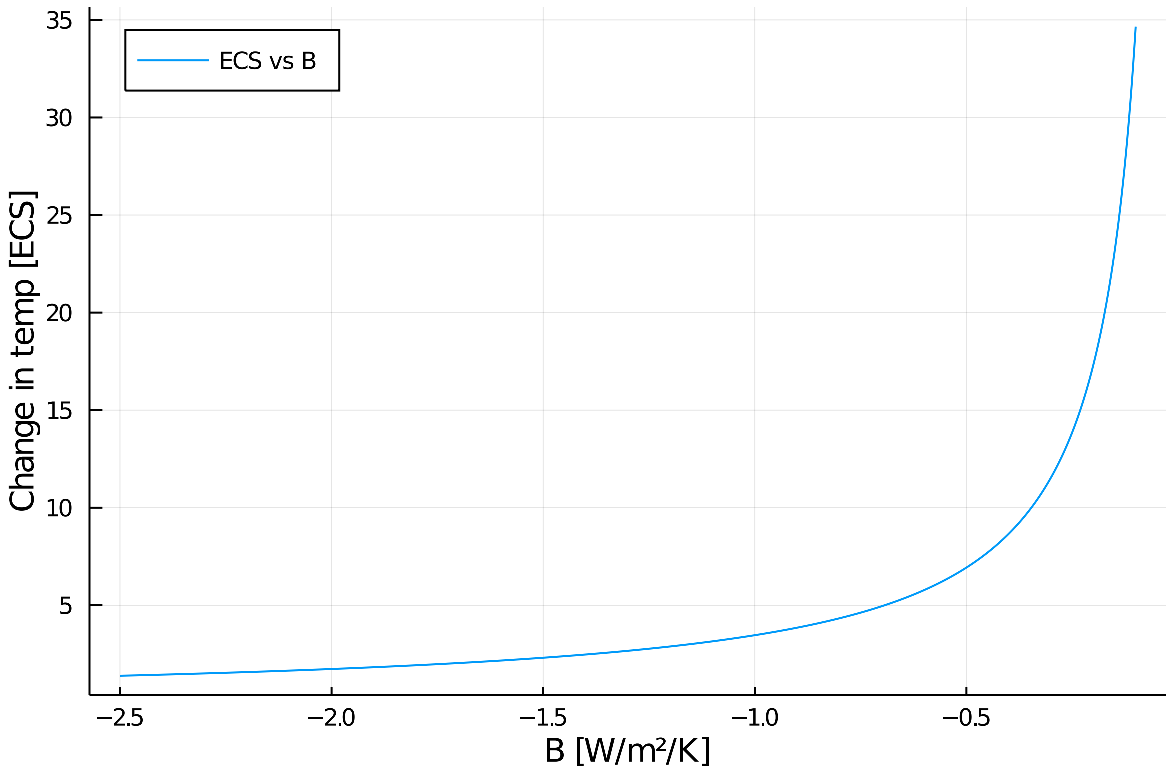
As soon as B goes beyong 0 , the whole model turns into a unstable point, with positive feedback making it grow more if it tries to.

```
observations_from_nonnegative_B = md"""
```

```
As soon as B goes beyong 0 , the whole model turns into a unstable point, with
positive feedback making it grow more if it tries to.
"""
```

Reveal answer: ☐

👉 Create a graph to visualize ECS as a function of B.

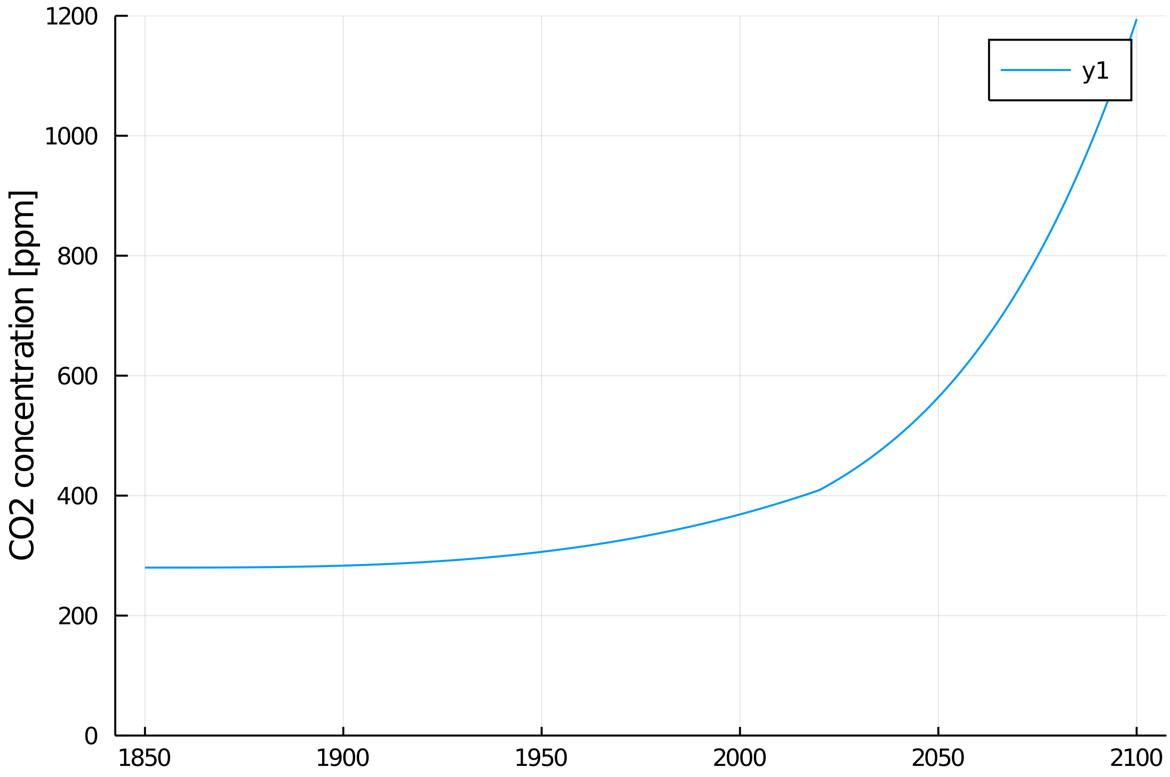


Exercise 1.2 - *Doubling CO_2*

To compute ECS, we doubled the CO_2 in our atmosphere. This factor 2 is not entirely arbitrary: without substantial effort to reduce CO_2 emissions, we are expected to **at least** double the CO_2 in our atmosphere by 2100.

Right now, our CO_2 concentration is 415 ppm – 1.482 times the pre-industrial value of 280 ppm from 1850.

The CO_2 concentrations in the *future* depend on human action. There are several models for future concentrations, which are formed by assuming different *policy scenarios*. A baseline model is RCP8.5 - a "worst-case" high-emissions scenario. In our notebook, this model is given as a function of t.




```
plot(t, Model.CO2_RCP85.(t),
    ylim=(0,1200), ylabel="CO2 concentration [ppm]")
```

👉 In what year are we expected to have doubled the CO_2 concentration, under policy scenario RCP8.5?

```
expected_double_CO2_year = 2050
```

```
expected_double_CO2_year = let

    t[findfirst(x-> x>2*Model.CO2_PI,Model.CO2_RCP85.(t))]
end
```

Hint

Exercise 1.3 - *Uncertainty in B*

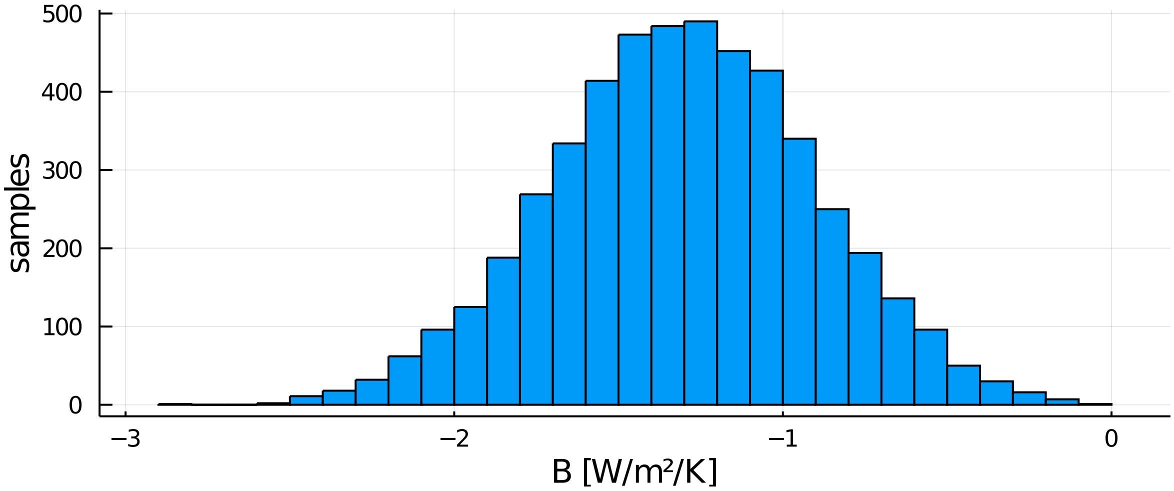
The climate feedback parameter B is not something that we can control– it is an emergent property of the global climate system. Unfortunately, B is also difficult to quantify empirically (the relevant processes are difficult or impossible to observe directly), so there remains uncertainty as to its exact value.

A value of B close to zero means that an increase in CO_2 concentrations will have a larger impact on global warming, and that more action is needed to stay below a maximum temperature. In answering such policy-related question, we need to take the uncertainty in B into account. In this exercise, we will do so using a Monte Carlo simulation: we generate a sample of values for B, and use these values in our analysis.

```
0.4
```

```
B̄ = -1.3; σ = 0.4
```

```
B_samples =
▸Float64[-1.45001, -2.39315, -0.991171, -1.16214, -1.35234, -1.44575, -1.72008, -1.(
```




```
histogram(B_samples, size=(600, 250), label=nothing, xlabel="B [W/m²/K]",
ylabel="samples")
```

👉 Generate a probability distribution for the ECS based on the probability distribution function for B above. Plot a histogram.

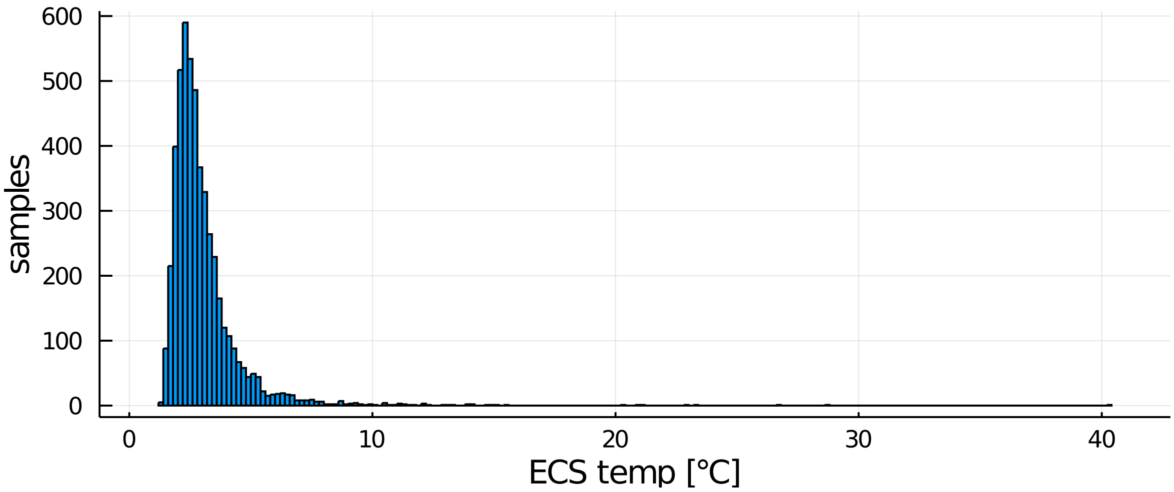
```
ECS_samples =
▸Float64[2.39014, 1.44819, 3.49661, 2.98221, 2.56276, 2.39719, 2.01487, 3.22547, 2.
```

```
ECS_samples = ECS.(;B=B_samples,a=Model.a)
```

Answer:

```
3.0351895924011676
```

```
mean(ECS_samples)
```




```
histogram(ECS_samples, size=(600, 250), label=nothing, xlabel="ECS temp [°C]",
ylabel="samples")
```

It looks like the ECS distribution is **not normally distributed**, even though B is.

👉 How does $\overline{\mathrm{ECS}(B)}$ compare to $\mathrm{ECS}(\overline{B})$? What is the probability that $\mathrm{ECS}(B)$ lies above $\mathrm{ECS}(\overline{B})$?

```
mean_ECS = 3.0351895924011676
  mean_ECS=mean(ECS_samples)
```

```
ECS_of_mean_B = 2.665950694461328
  ECS_of_mean_B=ECS(B=-1.3,a=Model.a)
```

```
probability = 0.49799919967987194
  probability= count(x->x>ECS_of_mean_B,ECS_samples)/length(ECS_samples)
```

ANSWER = We can cleary see mean of all ECS values is much more than ECS value of mean B, even though probability of of getting ECS value of a certain B more than ECS value of mean B is 1/2, which suggest ECS(B) function is a increasing func as seen by exercise 1.2

👉 Does accounting for uncertainty in feedbacks make our expectation of global warming better (less implied warming) or worse (more implied warming)?

`observations_from_the_order_of_averaging =`
It made more worse (implied warming) since,

$\overline{\mathrm{ECS}(B)} > \mathrm{ECS}(\overline{B})$

but getting a value above $\mathrm{ECS}(\overline{B})$ is as probable as getting below. but accounting for the extream large values of ECS when B touches -> 0, made it somewhat neither better nor worse.

Exercise 1.5 - *Running the model*

In the lecture notebook we introduced a *mutable struct* `EBM` (*energy balance model*), which contains:

- the parameters of our climate simulation (`C`, `a`, `A`, `B`, `CO2_PI`, `α`, `S`, see details below)
- a function `CO2`, which maps a time `t` to the concentrations at that year. For example, we use the function `t -> 280` to simulate a model with concentrations fixed at 280 ppm.

`EBM` also contains the simulation results, in two arrays:

- `T` is the array of tempartures (°C, `Float64`).
- `t` is the array of timestamps (years, `Float64`), of the same size as `T`.

Properties of an `EBM` obect:

Name	Description
`A`	Linearized outgoing thermal radiation: offset [W/m²]
`B`	Linearized outgoing thermal radiation: slope. *or:* **climate feedback parameter** [W/m²/°C]
`α`	Planet albedo, 0.0-1.0 [unitless]

Name	Description
S	Solar insulation [W/m²]
C	Atmosphere and upper-ocean heat capacity [J/m²/°C]
a	CO_2 forcing effect [W/m²]
CO2_PI	Pre-industrial CO_2 concentration [ppm]

You can set up an instance of `EBM` like so:

```
empty_ebm =
▶Main.workspace1359.Model.EBM(Float64[14.0], Float64[1850.0], 1.0, #7, 51.0, 5.0, 22
```

```
empty_ebm = Model.EBM(
    14.0, # initial temperature
    1850, # initial year
    1, # Δt
    t -> 280.0, # CO2 function
)
```

Have look inside this object. We see that `T` and `t` are initialized to a 1-element array.

Let's run our model:

```
simulated_model =
▶Main.workspace1359.Model.EBM(Float64[14.0, 14.0, 14.0, 14.0, 14.0, 14.0, 14.0, 14.(
```

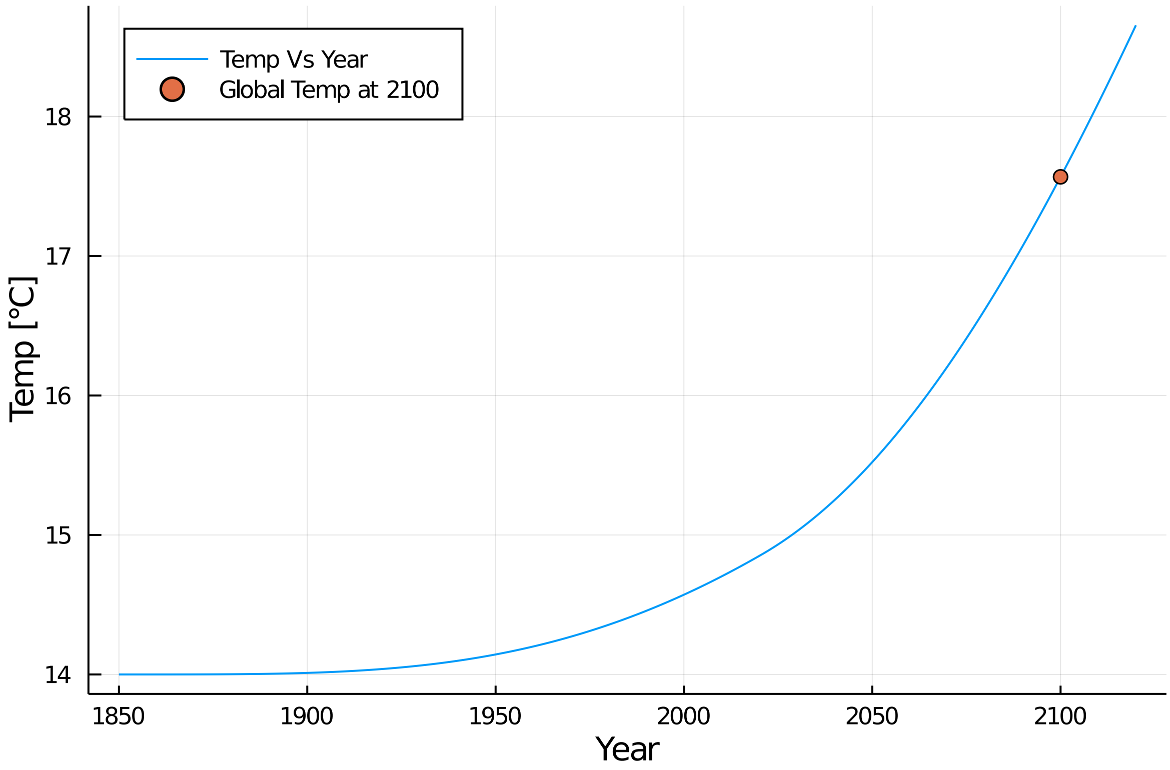
```
simulated_model = let
    ebm = Model.EBM(14.0, 1850, 1, t -> 280.0)
    Model.run!(ebm, 2020)
    ebm
end
```

Again, look inside `simulated_model` and notice that `T` and `t` have accumulated the simulation results.

In this simulation, we used `T0 = 14` and `CO2 = t -> 280`, which is why `T` is constant during our simulation. These parameters are the default, pre-industrial values, and our model is based on this equilibrium.

👉 Run a simulation with policy scenario RCP8.5, and plot the computed temperature graph. What is the global temperature at 2100?

```
simulated_rcp85_model =
```




```
simulated_rcp85_model = let

    ebm = Model.EBM(14.0,1850,1,Model.CO2_RCP85)
    Model.run!(ebm,2120)
    p=plot(ebm.t,ebm.T,xlabel="Year",ylabel="Temp [°C]",label="Temp Vs Year")
    scatter!(p,[(ebm.t[end-20],ebm.T[end-20])],label="Global Temp at 
2100",legend=:topleft)

end
```

Additional parameters can be set using keyword arguments. For example:

```
Model.EBM(14, 1850, 1, t -> 280.0; B=-2.0)
```

Creates the same model as before, but with `B = -2.0`.

👉 Write a function `temperature_response` that takes a function `CO2` and an optional value `B` as parameters, and returns the temperature at 2100 according to our model.

```
temperature_response (generic function with 2 methods)
```

```
function temperature_response(CO2::Function, B::Float64=-1.3)
    ebm = Model.EBM(14, 1850, 1, CO2; B=B)
    Model.run!(ebm,2100)
    return ebm.T[end]
end
```

```
14.0
```

```
temperature_response(t -> 280)
```

```
17.567568380496546
```

```
temperature_response(Model.CO2_RCP85)
```

```
22.311013974496277
```

```
temperature_response(Model.CO2_RCP85, -1.0)
```

Exercise 1.6 - *Application to policy relevant questions*

We talked about two *emissions scenarios*: RCP2.6 (strong mitigation - controlled CO2 concentrations) and RCP8.5 (no mitigation - high CO2 concentrations). These are given by the following functions:

```
▸(280.0,  280.0)
```

```
Model.CO2_RCP26(t_scenario_test), Model.CO2_RCP85(t_scenario_test)
```

1850

```
@bind t_scenario_test Slider(t; show_value=true, default=1850)
```

```
t = 1850:2100
```

We are interested in how the **uncertainty in our input** B (the climate feedback paramter) *propagates* through our model to determine the **uncertainty in our output** $T(t)$, for a given emissions scenario. The goal of this exercise is to answer the following by using *Monte Carlo Simulation* for *uncertainty propagation*:

> 👉 What is the probability that we see more than 2°C of warming by 2100 under the low-emissions scenario RCP2.6? What about under the high-emissions scenario RCP8.5?

```
prob_RCP26 = 0.4737895158063225
```

```
prob_RCP26=count(x->x>16.0,temperature_response.
(Model.CO2_RCP26,B_samples))/length(B_samples)
```

```
prob_RCP85 = 0.6332533013205283
```

```
prob_RCP85=count(x->x>16.0,temperature_response.
(Model.CO2_RCP85,B_samples))/length(B_samples)
```

Exercise 2 - *How did Snowball Earth melt?*

In lecture 21 (see below), we discovered that increases in the brightness of the Sun are not sufficient to explain how Snowball Earth eventually melted.

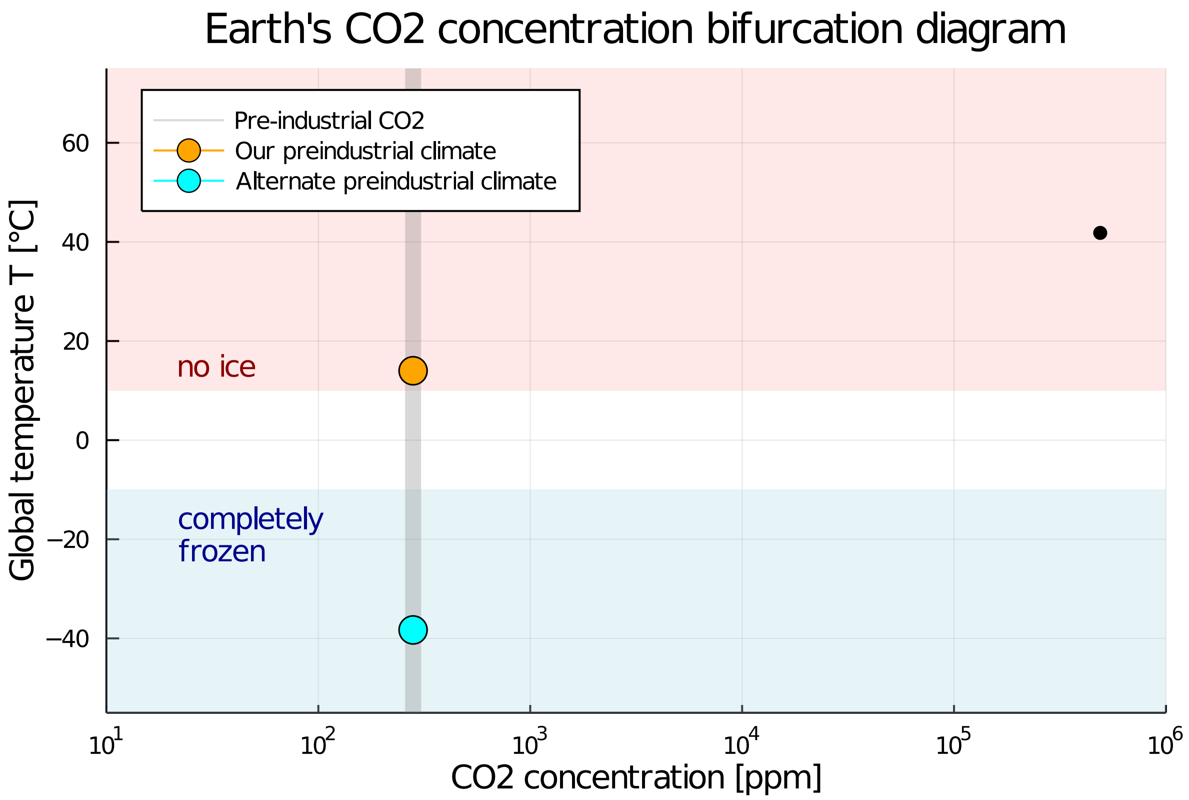
We talked about a second theory – a large increase in CO_2 (by volcanoes) could have caused a strong enough greenhouse effect to melt the Snowball. If we imagine that the CO_2 then decreased (e.g. by getting sequestered by the now liquid ocean), we might be able to explain how we transitioned from a hostile Snowball Earth to today's habitable "Waterball" Earth.

In this exercise, you will estimate how much CO_2 would be needed to melt the Snowball and visualize a possible trajectory for Earth's climate over the past 700 million years by making an interactive *bifurcation diagram*.

Exercise 2.1

In the **lecture notebook** (video above), we had a bifurcation diagram of S (solar insolation) vs T (temperature). We increased S, watched our point move right in the diagram until we found the tipping point. This time we will do the same, but we vary the CO_2 concentration, and keep S fixed at its default (present day) value.

Below we have an empty diagram, which is already set up with a CO_2 vs T diagram, with a logirthmic horizontal axis. Now it's your turn! We have written some pointers below to help you, but feel free to do it your own way.



We used two helper functions:

```
add_cold_hot_areas! (generic function with 1 method)
```

```
add_reference_points! (generic function with 1 method)
```

👉 Create a slider for `CO2` between `CO2min` and `CO2max`. Just like the horizontal axis of our plot, we want the slider to be *logarithmic*.

```
CO2 = 489778.81936844665
```

Hint

```
ebm =
▸Main.workspace1359.Model.EBM(Float64[-48.0],  Float64[0.0],  5.0,  CO2_const,  51.0,  5.0
```

```
ebm = Model.EBM(Tneo, 0., 5., Model.CO2_const)
```

👉 Write a function `step_model!` that takes an existing `ebm` and `new_CO2`, which performs a step of our interactive process:

- Reset the model by setting the `ebm.t` and `ebm.T` arrays to a single element. *Which value?*
- Assign a new function to `ebm.CO2`. *What function?*
- Run the model.

```
step_model! (generic function with 1 method)
```

```
function step_model!(ebm::Model.EBM, CO2::Real)

    # your code here
    ebm.t=[0.]
    ebm.T=[ebm.T[end]]
    ebm.CO2=x->CO2
    Model.run!(ebm)

    return ebm
end
```

👉 Inside the plot cell, call the function `step_model!`.

Parameters

```
CO2min = 10
```

```
CO2max = 1000000
```

```
Tneo = -48
```

The albedo feedback is implemented by the methods below:

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

```
function α(T; α0=Model.α, αi=0.5, ΔT=10.)
    if T < -ΔT
        return αi
    elseif -ΔT <= T < ΔT
        return αi + (α0-αi)*(T+ΔT)/(2ΔT)
    elseif T >= ΔT
        return α0
    end
end
```

```
function Model.timestep!(ebm)
    ebm.α = α(ebm.T[end]) # Added this line
    append!(ebm.T, ebm.T[end] + ebm.Δt*Model.tendency(ebm));
    append!(ebm.t, ebm.t[end] + ebm.Δt);
end
```

If you like, make the visualization more informative! Like in the lecture notebook, you could add a trail behind the black dot, or you could plot the stable and unstable branches. It's up to you!

Exercise 2.2

👉 Find the **lowest CO_2 concentration** necessary to melt the Snowball, programatically.

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

```
function equi(CO2::Int64)
    ebm = Model.EBM(Tneo, 0., 5., x -> CO2)
```

```
    Model.run!(ebm,500)
    ebm.T[end]
end
```

co2_to_melt_snowball = 478380

Hint

Hint

Exercise XX: *Lecture transcript*

(MIT students only)

Please see the link for hw 9 transcript document on **Canvas**. We want each of you to correct about 500 lines, but don't spend more than 20 minutes on it. See the the beginning of the document for more instructions. :point_right: Please mention the name of the video(s) and the line ranges you edited:

lines_i_edited =
Abstraction, lines 1-219; Array Basics, lines 1-137; Course Intro, lines 1-144 (*for example*)

```
lines_i_edited = md"""
Abstraction, lines 1-219; Array Basics, lines 1-137; Course Intro, lines 1-144 (_for example_)
"""
```

Function library

Just some helper functions used in the notebook.

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

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

```
still_missing (generic function with 2 methods)
```

```
keep_working (generic function with 2 methods)
```

```
yays =
▸Markdown.MD[Fantastic!, Splendid!, Great!, Yay ❤, Great! 🎉, Well done!, Keep it up
```

```
correct (generic function with 2 methods)
```

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

```
TODO =
```

TODO