Table of Contents

1 Code Präambel (bitte runterscrollen zum Hauptteil)

```
(ns mariastefan
  (:refer-clojure :exclude [+ - * / = abs compare zero? ref partial
                            numerator denominator infinite?])
  (:require [emmy.env :as e :refer :all :exclude [F->C]]
            [scicloj.kindly.v4.api :as kindly]
            [scicloj.kindly.v4.kind :as kind]))
```

```
(def html kind/hiccup)
```

```
(defn is-equal! [a b]
  (when (zero? (simplify (- a b))) a))
```

```
(defn solves! [a f]
  (when (zero? (simplify (f a))) a))
```

```
(defn postfix? [ctx ex]
  ((-> ctx :Schlusselworte :postfix)
   (str (last ex))))
```

```
(defn infix? [ctx ex]
  ((-> ctx :Schlusselworte :infix)
   (str (second ex))))
```

```
(defn infix-function? [ctx ex]
  ((-> ctx :Schlusselworte :infix-function) (str (second ex))))
```

```
(defn ifx-fn-mp-decon? [ctx ex]
  (and (infix-function? ctx ex)
       ((-> ctx :Schlusselworte :infix-function-map-deconstrucion)
        (str (second (last ex))))))
```

```
(defn ifx-fn-reverse? [ctx ex]
  (and (infix-function? ctx ex)
       ((-> ctx :Schlusselworte :infix-function-reverse)
        (str (second (last ex))))))
```

```
(defn notext? [ctx smb]
  ((-> ctx :Schlusselworte :notext)
   (str smb)))
```

```
(defn npow [x n]
  (apply * (repeat n x)))
```

```
(defn mypow [x n]
  (if (integer? n)
    (npow x n)
    (exp (* (log x) n))))
```

```
(defn n-transpose [m]
  (apply mapv vector m))
```

```
(defn r-transpose [x]
  (reverse (n-transpose x)))
```

```
(defn mround [x] (floor (+ x 0.5)))
```

```
(defn to-double [x]
  (let [n (exp -30)]
    (- (+ x n) n)))
```

```
(defn round [x dec]
  (to-double (/ (mround (* x (mypow 10 dec))) (mypow 10 dec))))
```

```
(defn str-to-int [i]
  ;; in maria.cloud
  #_(int i)
  (Integer/parseInt i))
```

```clojure
(def plus +)
```

```clojure
(def minus -)
```

```clojure
(def mal *)
```

```clojure
(def von *)
```

```clojure
(def dot *)
```

```clojure
(def durch /)
```

```clojure
(def hoch mypow)
```

```clojure
(def null 0)
```

```clojure
(def Ein 1)
```

```clojure
(def Milliard (fn [x] (* x (mypow 10 9))))
```

```clojure
(def stel (fn [x y] (/ x y)))
```

```clojure
(defn Komma [& lst]
  (let [a (apply str (rest lst))]
    (to-double (+ (first lst) (/ (str-to-int a) (mypow 10 (count a)))))))
```

```clojure
(defn sw [ctx ex]
  (cond
    (not (coll? ex)) ex
    (ifx-fn-mp-decon? ctx ex)
    (list (list 'fn [{:keys (first (last ex))}] (first ex))
          (last (last ex)))
    (ifx-fn-reverse? ctx ex)
    (list (list 'fn [(last (last ex))] (first ex))
          (first (last ex)))
    (infix-function? ctx ex)
    (list (list 'fn [(first (last ex))] (first ex))
          (last (last ex)))
    (postfix? ctx ex)
```

```
    (conj (butlast ex) (last ex))
    (infix? ctx ex)
    (conj (rest (rest ex)) (first ex) (second ex))
    :else            ex))
```

```
(defn bx [ctx ex]
  (if (not (coll? ex)) ex
      (str "\\fbox{"
           (apply str
                  (interpose " " (remove #(notext? ctx %) ex)))
           "}")))
```

```
(defn maxcount [xs mini]
  (apply max (conj (map #(-> (second %) :style :count) (filter coll? xs)) mini)))
```

```
(defn b [p] {:style {:border "1px solid gray" :padding (str p "px") :count p}})
```

```
(defn hx [ctx ex]
  (if (not (coll? ex))
    (str " " ex " ")
    (let [rex (remove #(notext? ctx %) ex)]
      (into [:span (b (+ 3 (maxcount rex 2)))] rex))))
```

```
(defn table-formula? [ctx ex]
  (and (coll? (last ex))
       ((-> ctx :Schlusselworte :infix-function) (str (first (last ex))))))
```

```
(defn threadzero [ctx ex]
  (if (table-formula? ctx ex)
    (if (> (count ex) 2)
      (cons (threadzero ctx (butlast ex)) (last ex))
      (cons (first ex) (last ex)))
    ex))
```

```
(def gctx {:Schlusselworte
           {:infix
            #{"plus" "minus" "mal" "von" "dot" "durch" "hoch" "Ein" "Milliard"
              "Komma"}
            :postfix
            #{"stel"}
```

```
            :infix-function
            #{"mit" "und"}
            :infix-function-map-deconstrucion
            #{"aus"}
            :infix-function-reverse
            #{"für"}
            :notext
            #{"dot"}}
           :Schulwissen
           {:e  euler
            :pi pi}})
```

```
(defn my-walk
  [inner outer form]
  (cond
    (list? form)      (outer (apply list (map inner form)))
    (map-entry? form) form
    (seq? form)       (outer (doall (map inner form)))
    (record? form)    (outer (reduce (fn [r x] (conj r (inner x))) form form))
    (coll? form)      (outer (into (empty form) (map inner form)))
    :else             (outer form)))
```

```
(defn postwalk
  [f form]
  (my-walk (partial postwalk f) f form))
```

```
(defmacro calcbox [ex & dbg]
  (let [swe (postwalk #(sw gctx %) (threadzero gctx ex))
        bxe (if (vector? ex)
              (->> (cons (first ex) (apply concat (rest ex)))
                   (map (fn [mex] (postwalk #(bx gctx %) mex)))
                   (interpose " \\\\ ")
                   (apply str))
              (postwalk #(bx gctx %) ex))
        hxe (if (vector? ex)
              (->> (cons (first ex) (apply concat (rest ex)))
                   (map (fn [mex] (postwalk #(hx gctx %) mex)))
                   (map #(vector :p [:div %]))
                   (into [:p]))
              [:p (postwalk #(hx gctx %) ex)])]
    {:code   `'~swe
```

```
     :calc   (if (seq dbg) (into [] dbg) swe)
     :tex    bxe
     :hiccup hxe}))
```

2 J. Stefan: Über die Beziehung zwischen der Wärmestrahlung und der Temperatur

Temperatur in Celsius

```
(def temper [80 100 120 140 160 180 200 220 240])
```

Dulong und Petit angegebene Messdaten (extrapoliert aus Messungen)

```
(def dp-meas [1.74 2.30 3.02 3.88 4.89 6.10 7.40 8.81 10.69])
```

von D&P angegeben als nach ihrer Heuristik gerechnet

```
(def dp-theor [1.72 2.33 3.05 3.89 4.87 6.03 7.34 8.89 10.68])
```

vom Stefan berechnete Werte

```
(def st-theor [1.66 2.30 3.05 3.92 4.93 6.09 7.42 8.92 10.62])
```

```
(defn dp-formel [Celsius]
  (calcbox [((((A hoch x) minus 1) mal M)
             mit
             [x in Celsius])
            [und [(1 Komma 0 0 77) für A]]
            [und [(2 Komma 0 2) für M]]]))
```

```
(html (:hiccup (dp-formel 0)))
```

```
(def tb2 (map (fn [[T m t]]
                [T m (round (- t m) 2) t (round (:calc (dp-formel T)) 2)])
              (n-transpose [temper dp-meas dp-theor])))
```

Temperatur Messwert Differenz RechenWert_D&P Meine-Rechnung

```
tb2
```

```
([80 1.74 -0.02 1.72 1.71]
 [100 2.3 0.03 2.33 2.33]
 [120 3.02 0.03 3.05 3.05]
 [140 3.88 0.01 3.89 3.89]
 [160 4.89 -0.02 4.87 4.87]
 [180 6.1 -0.07 6.03 6.01]
 [200 7.4 -0.06 7.34 7.35]
 [220 8.81 0.08 8.89 8.9]
 [240 10.69 -0.01 10.68 10.71])
```

```
(defn stefan-formel [Celsius]
  (calcbox [((((T plus x) hoch 4 )
              minus
              (T hoch 4))
             mal
             B)
            [mit [x in Celsius]]
            [und [(Ein (6 Milliard) stel) für B]]
            [und [273 für T]]]))
```

```
(html (:hiccup (stefan-formel 0)))
```

```
(def pow_273_4 (calcbox [(((T mal T) mal T) mal T) [mit [T gleich 273]]]))
```

```
(html (:hiccup pow_273_4))
```

```
(:calc pow_273_4)
```

```
5554571841
```

Unten passt die Differenz mit Paper überein, auch Meine-Rechnung lt. Paper gerechnet passt

```
(def tb7 (map (fn [[T m st-t]]
                [T m (round (- st-t m) 2) st-t
                 (round (:calc (stefan-formel T)) 2)])
              (n-transpose [temper dp-meas st-theor])))
```

Temperatur Messwert Differenz Rechenwert_St Meine-Rechnung

```
tb7
```

```
([80 1.74 -0.08 1.66 1.66]
 [100 2.3 0.0 2.3 2.3]
 [120 3.02 0.03 3.05 3.05]
 [140 3.88 0.04 3.92 3.92]
 [160 4.89 0.04 4.93 4.93]
 [180 6.1 -0.01 6.09 6.09]
 [200 7.4 0.02 7.42 7.42]
 [220 8.81 0.11 8.92 8.92]
 [240 10.69 -0.07 10.62 10.62])
```

```
(defn minsec [sec]
  (mapv mround [(quot sec 60) (mod sec 60)]))
```

```
(defn speed->sec [sp]
  (* (/ 20.0 sp) 60))
```

```
(defn tb3 [tbl]
  (map (fn [tb]
         (map #(minsec (speed->sec %)) tb))
       tbl))
```

Umrechnung von Messwerten und Rechnungen nach Zeitdauer – MinutenSekunden, Absteigende Temperatur

```
(tb3 (r-transpose [dp-meas dp-theor st-theor]))
```

```
(([1 52] [1 52] [1 53])
 ([2 16] [2 15] [2 15])
 ([2 42] [2 43] [2 42])
 ([3 17] [3 19] [3 17])
 ([4 5] [4 6] [4 3])
 ([5 9] [5 8] [5 6])
 ([6 37] [6 33] [6 33])
 ([8 42] [8 35] [8 42])
 ([11 30] [11 38] [12 3]))
```

```
(def Minuten 'Minuten)
```

```
(def Minute 'Minute)
```

```
(def Sekunden 'Sekunden)
```

```
(def Sekunde 'Sekunde)
```

```
(def Eine 'Eine)
```

```
(def und 'und)
```

```
(def ist 'ist)
```

```
(def Null 'Null)
```

```
(defn text-minute [m]
  (case m
    0 [Null Minuten]
    1 [Eine Minute]
    [m Minuten]))
```

```
(defn text-sekunde [m]
  (case m
    0 [Null Sekunden]
    1 [Eine Sekunde]
    [m Sekunden]))
```

```
(defn tb4 [tb]
  (mapv (fn [[m s]] [(text-minute m) und (text-sekunde s)])
        tb))
```

Zeitdauer der Messwerte in Worten

```
(tb4 (map first (tb3 (r-transpose [dp-meas]))))
```

```
[[[Eine Minute] und [52 Sekunden]]
 [[2 Minuten] und [16 Sekunden]]
 [[2 Minuten] und [42 Sekunden]]
 [[3 Minuten] und [17 Sekunden]]
 [[4 Minuten] und [5 Sekunden]]
```

```
 [[5 Minuten] und [9 Sekunden]]
 [[6 Minuten] und [37 Sekunden]]
 [[8 Minuten] und [42 Sekunden]]
 [[11 Minuten] und [30 Sekunden]]]
```

```
(def minus- 'minus)
```

```
(def plus+ 'plus)
```

```
(defn tb6 [tb]
  (mapv (fn [[[am as] [bm bs]]]
          [[bs
            (if (neg? (- as bs)) minus- plus+)
            (abs (- as bs))]
           ist
           as])
        tb))
```

Differenz der Sekunden: D&P-Rechnung Differenz D&P-Messung

```
(tb6 (tb3 (r-transpose [dp-meas dp-theor])))
```

```
[[[52 plus 0] ist 52]
 [[15 plus 1] ist 16]
 [[43 minus 1] ist 42]
 [[19 minus 2] ist 17]
 [[6 minus 1] ist 5]
 [[8 plus 1] ist 9]
 [[33 plus 4] ist 37]
 [[35 plus 7] ist 42]
 [[38 minus 8] ist 30]]
```

Differenz der Sekunden: St-Rechnung Differenz D&P-Messung

```
(tb6 (tb3 (r-transpose [dp-meas st-theor])))
```

```
[[[53 minus 1] ist 52]
 [[15 plus 1] ist 16]
 [[42 plus 0] ist 42]
 [[17 plus 0] ist 17]
```

```
[[3 plus 2] ist 5]
[[6 plus 3] ist 9]
[[33 plus 4] ist 37]
[[42 plus 0] ist 42]
[[3 plus 27] ist 30]]
```

3 Kleine Fingerübungen

```
(def bsp0 (calcbox (1 plus 3)))
```

```
(html (:hiccup bsp0))
```

```
(:calc bsp0)
```

```
4
```

praktisch zum debuggen: der generierte code

```
(:code bsp0)
```

```
(plus 1 3)
```

```
(def bsp1 (calcbox (2 plus 1) "andere Rechnung" (+ 4 5)))
```

```
(html (:hiccup bsp1))
```

noch praktischer zum debuggen: calculation unterdrücken

```
(:calc bsp1)
```

```
["andere Rechnung" 9]
```

```
(def bsp2 (calcbox ((X plus 1) mit [X gleich [(Y plus 2) mit [Y gle=ich 3]]])))
```

```
(html (:hiccup bsp2))
```

```
(:calc bsp2)
```

```
6
```

```
(def bsp3 (calcbox ( [(X plus Y) mit [X ist-gleich 3]] mit [Y ist 2]) ))
```

es ist wurst ob glei=ch, ist-gleich, -in-: nur ein Füllwort

```
(html (:hiccup bsp3))
```

```
(:calc bsp3)
```

```
5
```

```
(defn bsp4 [Sekunden Y]
  (calcbox ((X plus Y) mit [X -in- Sekunden])))
```

```
(html (:hiccup (bsp4 0 0)))
```

```
(:calc (bsp4 5 6))
```

```
11
```

```
(defn bsp5 [{:keys [Schulwissen]}]
  (calcbox ((e hoch pi) mit [[e pi] aus Schulwissen])))
```

```
(html (:hiccup (bsp5 gctx)))
```

```
(def bsp6 (calcbox [(X plus Y) [mit [X gleich (Y plus 7)]] [mit [Y gle=ich 3]]]))
```

```
(html (:hiccup bsp6))
```

4 deBroglie Wavelength

Internal Vibarations

as always with Einstein, we start with E = mc^2

```
(defn E0 [m] (* m 'c 'c))
```

deBroglies first hypothesis was to assume that every particle has a hypothetical internal vibration at frequency nu0 which relates to the rest energy in rest frame of particle (only there this energy-frequency relation holds)

```
(defn nu_naught [E0] (/ E0 'h))
```

particle travels at velocity v

```
(defn v [beta] (* beta 'c))
```

```
(defn beta [v] (/ v 'c))
```

```
(defn gamma [beta] (/ 1 (sqrt (- 1 (* beta beta)))))
```

time dilation: internal vibration is slower for observer. so the frequency-energy relation does not hold: the frequency indeed decreases instead of increasing with energy. this is the conundrum deBroglie solved. so hang on.

```
(defn nu_one [nu_naught gamma] (/ nu_naught gamma))
```

sine formula for internal vibration. we do not know what exactly vibrates so we set the amplitude to one

```
(defn internal-swing [nu_one]
  (fn [t] (sin (* 2 'pi nu_one t))))
```

calculate the phase of the internal swing at particle point x = v * t

```
(defn internal-phase [nu_one x v]
  (asin ((internal-swing nu_one) (/ x v))))
```

```
(is-equal! (* 2 'pi 'nu_one (/ 'x 'v))
           (internal-phase 'nu_one 'x 'v))
```

```
(* 2 pi nu_one (/ x v))
```

personal note: to me, this is the sine-part of a standing wave, the standing vibration.

A general Wave

now for something completely different: general definition of a wave

```
(defn wave [omega k]
  (fn [x t] (sin (- (* omega t) (* k x)))))
```

with the usual definition of omega

```
(defn omega [nu] (* 2 'pi nu))
```

and the simplest possible definition for the wave-vector k: a dispersion free wave traveling at phase-velocity V

```
(defn k [omega V] (/ omega V))
```

calculate the phase of the wave

```
(defn wave-phase [nu x t V]
  (asin ((wave (omega nu) (k (omega nu) V)) x t)))
```

```
(is-equal! (* 2 'pi 'nu (- 't (/ 'x 'V)))
           (wave-phase 'nu 'x 't 'V))
```

```
(* 2 pi nu (- t (/ x V)))
```

Phase difference

calculate the phase difference between the vibration and some wave at time t = x / v as a function of the ratio of the frequencies

```
(defn phase-difference [r x v nu V]
  (- (internal-phase (* r nu) x v)
     (wave-phase nu x (/ x v) V)))
```

```
(is-equal! (* 2 'pi 'nu (+ (* (- 'r 1) (/ 'x 'v)) (/ 'x 'V)))
           (phase-difference 'r 'x 'v 'nu 'V))
```

```
(* 2 pi nu (+ (* (- r 1) (/ x v)) (/ x V)))
```

state the general ratio of frequencies that keeps the vabration of the particle in phase with some wave of velocity V in terms of the velocity of the particle

```
(solves! (@(defn nu-ratio-in-phase [v V] (- 1 (/ v V)))
          'v 'V)
         (fn [r] (phase-difference r 'x 'v 'nu 'V)))
```

```
(- 1 (/ v V))
```

the Energy of the particle for the observer

```
(defn E [E0 gamma] (* E0 gamma))
```

we assume the deBroglie wave has the frequency: energy devided by Planck's constant. reminder: this relation holds in every frame of reference, especially for the observer who is not in the rest frame.

```
(defn nu [E] (/ E 'h))
```

now that nu is set, calculate the physically viable ratio of the frequencies in terms of beta

```
(defn physical-nu-ratio [beta]
  (/ (nu_one (nu_naught 'E0) (gamma beta))
     (nu (E 'E0 (gamma beta)))))
```

```
(is-equal! (- 1 (* 'beta 'beta))
           (physical-nu-ratio 'beta))
```

```
(- 1 (* beta beta))
```

state the value of the physical phase-velocity V that keeps the vibration and the deBroglie wave in phase in terms of the particle velocity v

```
(solves! (@(defn phase-velocity [beta] (/ 'c beta))
           'beta)
         (fn [V] (- (physical-nu-ratio 'beta)
                    (nu-ratio-in-phase (v 'beta) V))))
```

```
(/ c beta)
```

note: the phase-velocity is always greater than the speed of light. It is independent of the position x and the mass of the particle

the relativistic momentum is defined as

```
(defn p [m v gamma]
  (* m v gamma))
```

calculate the deBroglie wavelength (by dividing the phase-velocity by the frequency) and show that it indeed is h devided by the momentum

```
(def de-broglie-wavelength
  (/ (phase-velocity (beta 'v))
     (nu (E (E0 'm) 'gamma))))
```

```
(is-equal! (/ 'h (p 'm 'v 'gamma))
           de-broglie-wavelength)
```

```
(/ h (* m v gamma))
```

personal note: one can see this upside down. V and nu define not only the deBroglie phase wave but also a standing wave (standing vibration). and the intersection of the two waves gives the trajectory of the particle (and hence it's velocity v). Intersection meaning the points where the phase of the wave and the sine-part of the standing vibration-wave have the same value. The mass of the particle is then given by the deBroglie-wavelength and the v. Mass is thus a constant of the motion, the same value in every frame.

```
(println "Success!!")
```

```
nil
```