**APPENDIX 1**

**The Function of “Correcting Reconstructions of Moisture Index (MI) for [CO2]” in Python**

**A. Definition of the Function “calculate\_m\_true”**

class P\_model\_inverter:

 # Define various constants

 omega = 3

 lam = 2.45 # Latent heat of vaporisation (MJ kg^-1)

 gamma = 0.067 # psychrometer constant (kPa K^-1)

 a = 0.6108

 b = 17.27

 c = 237.3

 abc = 2503.1628468

 dark\_scale = 0.025

 visc\_offset = 138

 R\_o = 400

 R = 8.314 # Universal gas constant (J mol^-1 K^-1)

 dHc = 79430 # Carbon activation energy (J mol^-1)

 dHo = 36380 # Oxygen activation energy (J mol^-1)

 delta\_H = 37830 # Compensation point activation energy (J mol^-1)

 O = 210 # Atsmopheric concentration of oxygen

 C = 14.76

 modern\_CO2 = 340 # Modern CO2 concentration in ppm

 D\_root\_factor = 4

 solver\_args = None

 # Initalise given the reference variables

 def \_\_init\_\_(self, T\_diff, T\_ref, m\_rec, c\_ratio, lat=-30, \*args):

 # Set given values

 self.solver\_args = args

 self.lat = lat

 self.T\_rec = T\_diff + T\_ref

 self.T\_ref = T\_ref

 self.m\_rec = m\_rec

 self.c\_ratio = c\_ratio

 # Precalculate some variables that don't change for changing true MI

 self.K\_rec = K(self.T\_rec)

 self.K\_ref = K(self.T\_ref)

 self.eta\_rec = eta(self.T\_rec)

 self.eta\_ref = eta(self.T\_ref)

 self.E\_q\_sec\_rec = pre\_section\_E\_q(self.T\_rec)

 self.E\_q\_sec\_ref = pre\_section\_E\_q(self.T\_ref)

 self.use\_e\_pre = self.c\_ratio\*( self.useable\_e(self.T\_ref, self.m\_rec, self.K\_ref, self.eta\_ref, self.E\_q\_sec\_ref) )

 # Find the optimal MI given the set variables

 def solve\_for\_delta\_m(self):

 return abs(fsolve(self.e\_difference, 1, \*self.solver\_args)[0]) - self.m\_rec

 def e\_difference(self, m\_true):

 m\_true = abs(m\_true)

 return abs(self.useable\_e(self.T\_rec, m\_true, self.K\_rec, self.eta\_rec, self.E\_q\_sec\_rec) - self.use\_e\_pre)

 # The version of E without unnecessary constants

 def useable\_e(self, T, m, pre\_K, pre\_eta, pre\_sec\_E\_q):

 pre\_E\_q = pre\_calc\_E\_q(T,m, pre\_sec\_E\_q)

 cur\_eqm = abs(eqm(pre\_E\_q, m))

 return cur\_eqm \* (self.C\*pow(pre\_eta/pre\_K, 1/2) \* pow(cur\_eqm, 1/P\_model\_inverter.D\_root\_factor) + 1) \*\*(-1)

 # Internal c\_i for the plant

 def get\_c\_i(self):

 return c\_i(self.T\_rec, self.c\_ratio\*self.modern\_CO2, self.m\_rec)

 # Determines if the compensation point 'law' is upheld

 def compensation\_point\_held(self, m):

 return c\_i(self.T\_rec, self.c\_ratio\*self.modern\_CO2, m) > true\_compensation\_point(self.T\_rec)

def eqm(pre\_E\_q, m):

 return pre\_E\_q \* ( pow(abs(1 + pow(m, P\_model\_inverter.omega)), 1/P\_model\_inverter.omega) - m )

def E\_q(T, m):

 return R\_n(T,m)/P\_model\_inverter.lam \* pow(1 + P\_model\_inverter.gamma \* pow(P\_model\_inverter.c+T, 2)/(P\_model\_inverter.abc) \* np.exp(-P\_model\_inverter.b\*T/(P\_model\_inverter.c + T)),-1)

def pre\_calc\_E\_q(T, m, pre):

 return R\_n(T, m)\*pre

def pre\_section\_E\_q(T):

 return pow(1 + P\_model\_inverter.gamma \* pow(P\_model\_inverter.c+T, 2)/(P\_model\_inverter.abc) \* np.exp(-P\_model\_inverter.b\*T/(P\_model\_inverter.c + T)),-1)/P\_model\_inverter.lam

# Gives the value for R\_n in MJ kg^(-1) a^(-1) mm

def R\_n(T, m, lat=-30):

 scale\_factor = 365.24\*24\*60\*60\*10\*\*(-6)

 return scale\_factor\*(0.83\*P\_model\_inverter.R\_o\*(0.25 + 0.5\*S\_f(m)) - (107 - T)\*(0.2 + 0.8\*S\_f(m)))

# Fraction of sunshine hours

def S\_f(m):

 return 0.6611 \* np.exp(-0.74\*m) + 0.2175

def K(T):

 pre\_calc = 1/P\_model\_inverter.R\*(1/298 - 1/(T + 273.15))

 return 404.9 \* np.exp(P\_model\_inverter.dHc \* pre\_calc) \* ( 1 + P\_model\_inverter.O/(278.4 \* np.exp(P\_model\_inverter.dHo \* pre\_calc)))

def eta(T):

 return 0.024258 \* np.exp(580/(T+P\_model\_inverter.visc\_offset))

# Calculates c\_i, could be sped up

def c\_i(T, c\_a, m):

 return c\_a/c\_a\_c\_i\_ratio(T, m)

def c\_a\_c\_i\_ratio(T, m):

 delta\_e = eqm(E\_q(T,m), m)

 return 1+ P\_model\_inverter.C\*(eta(T)/K(T))\*\*(1/2) \* delta\_e\*\*(1/P\_model\_inverter.D\_root\_factor)

def compensation\_point(T):

 return 42.75\*np.exp((P\_model\_inverter.delta\_H/P\_model\_inverter.R)\*(1/298 - 1/(273.15 + T)))

def true\_compensation\_point(T):

 return compensation\_point(T) + P\_model\_inverter.dark\_scale\*K(T)

# Budyko relationship to give alpha from mi

def alpha\_from\_mi\_om3(mi):

 return 1 + mi - (1+mi\*\*3)\*\*(1/3)

def calculate\_m\_true(T\_diff, T\_ref, m\_rec, c\_a\_diff):

 model = P\_model\_inverter(T\_diff, T\_ref, m\_rec, c\_a\_diff)

 delta\_m = model.solve\_for\_delta\_m()

 comp\_point\_held = model.compensation\_point\_held(m\_rec)

 output = m\_rec

 if comp\_point\_held:

 output += delta\_m

 return [output, comp\_point\_held, model.get\_c\_i()]

**B. Usage of the Function “calculate\_m\_true”**

**Input:**

past temperature – present\_temperature, present\_temperature, reconstructed\_mi, past\_c\_a/modern\_c\_a.

Past temperature (degree C): the past mean annual temperature derived from reconstructed MTCO and GDD0 for each sample;

Present\_temperature (degree C): the modern mean annual temperature;

Reconstructed \_mi (unitless): the reconstructed Moisture Index before correction for each sample;

Past\_c\_a (ppm): the past CO2 concentration for each sample;

Modern\_c\_a (ppm): the modern CO2 concentration, assumed to be 340 ppm in this model.

**Output:**

corrected\_moisture\_index, comp\_point\_held, model.get\_c\_i()

 Corrected\_moisture\_index (unitless): the moisture index after CO2 correction;

 Comp\_point\_held: test if the compensation point has been held;

 Model.get\_c\_i() (ppm): internal c\_i for original values.

**Usage example:**

**Input:**

# present\_temp = 20

# past\_temp = 15

# reconstructed\_mi = 1.2

# modern\_c\_a = P\_model\_inverter.modern\_CO2

# past\_c\_a = 250.0

# result = calculate\_m\_true(past\_temp - present\_temp, present\_temp, reconstructed\_mi, past\_c\_a/modern\_c\_a)

# print("New moisture index: ", result[0])

# print("Has the compensation point held?: ", result[1])

# print("Internal c\_i for original values: ", result[2])

**Output:**

New moisture index: 1.2037778166331266

Has the compensation point held?: True

Internal c\_i for original values: 59.750310167634716

**APPENDIX 2**

**An Approximation of the Dependence of GDD0 (Growing Degree Days on a Zero Base) on *T*min (Coldest-Month Temperature), *T*0 (Mean Annual Temperature) and *Tmax* (Warmest-Month Temperature).**

Assume that temperature (T) follows a sinusoidal pattern with time of year:

*T* = *T*0 – Δ*T* cos θ

where Δ*Τ* is the half-amplitude of seasonal variation in temperature, and θ is the ‘day angle’:

θ = (2π/365) *t*d

where *t*d­ is the day of the year, measured from a starting point in midwinter. Then:

*T*min = T0 – Δ*T*

GDD0 = ∫T>0 (*T*0 – Δ*T* cos θ) dθ (in units of K rad, where 1 K rad = 2π/365 K day)

The day-angle when T = 0 is given by θ0 = cos–1 (*T*0/Δ*T*), except in two special cases:

1) *T*0 > Δ*T* => GDD0 = 2π*T­*0 (this is the case when *T*min ≥ 0, hence every day is a growing day)

2) –*T*0 > Δ*T* => GDD0 = 0 (this is the case when there are no growing days)

Otherwise:

GDD0 = 2 [ *T*0 θ – Δ*T* sin θ ] evaluated from θ0 to π

= 2 π *T*0 – 2 *T*0 cos–1 (*T*0/Δ*T*) ­– 2 Δ*T* [sin π – sin cos–1 (*T*0/Δ*T*)]

= 2 π *T*0 – 2 *T*0 cos–1 (*T*0/Δ*T*) – 2 Δ*T* √[1 – (*T*0/Δ*T*)2]

Write *u = T*0/Δ*T*

Then GDD0 = 2 Δ*T* [π *u* – *u* cos–1 *u* + √(1 – *u*2)]

 = 2 Δ*T* [*u* cos–1 (–*u*) + √(1 – *u*2)].

This is in units of K rad. Multiplication by 365/2π converts this to units of K day.

**Predicting *T*0 and *Tmax* from GDD0 and *T*min**

From the logic above:

– *T*min/Δ*T* = 1 – *u* *and* GDD0/ΔT = 2 [*u* cos–1 (–*u*) + √(1 – *u*2)]

Therefore:

– GDD0 /*T*min = 2 [*u* cos–1 (–*u*) + √(1 – *u*2)] / (1 – *u*)

 = 2 {[(*u*/(1 – *u*)] cos–1 (–*u*) + √[(1 + *u*)/(1 ­– *u*)]}.

To estimate mean temperature (*T*0): convert GDD0 from K day to K rad, take the ratio of GDD0 to (– *T*min), and solve the equation above for *u*. Then,

*T*0 = – *T*min *u*/(1 – *u*)

Δ*T* = – *T*min /(1 – *u*)

and

*Tmax = T0* + Δ*T.*