A Brief Note on Climate Sensitivity

Thermodynamics is a funny subject. The first time you go through it, you don't understand it at all. The second time you go through it, you think you understand it, except for one or two small points. The third time you go through it, you know you don't understand it, but by that time you are so used to it, it doesn't bother you any more.

Arnold Sommerfeld

In an earlier note we proposed a thermodynamic algorithm for calculation of tropospheric thermal and energy flux profiles.[1] It was based on a previously derived dissipation theorem, [2]

> ***Theorem:*** *The thermodynamic steady state is that configuration, compatible with boundary constraints, requiring the minimum amount of work from external sources to counter relaxation towards equilibrium.*

Thermodynamic equilibrium is a steady-state of maximum entropy and zero dissipation. Non-equilibrium steady-states exhibit minimal dissipation but lack extremals for entropy formation, save in isothermal cases. In *Algebra I,* we learned of an equation describing the dissipation of an electric current flowing between two interfaces, $W=J\cdot(V_1-V_2)$. This is a most remarkable formula for it requires only boundary values and no information as to what lies betwixt. Equally 'trivial' expressions apply to fluxes of energy and mass. For the former, $W=(J/T_1)\cdot(T_1-T_2)$, some may recognize as Carnot's Equation. Should W be a minimum for steady-state models with T_1 and T_2 fixed , then so must be J. The internal profile, *T(x)*, then becomes a variational function when given a local function, *J(x)*. As a general form, we adopt $J(T)=f(x)\nabla T$ with x a normalized altitude $(0\le x\le 1)$. Thermodynamic models, by the very definition of temperature, contain no information of the path by which a steady state has been reached. Physical models, based on parameters of mass, distance and time, can predict, at least theoretically, states of evolution or devolution. In thermodynamic models the missing information appears as entropy.

Despite a half-century of atmospheric modeling, the relationship between temperatures and fluxes remains unresolved. To model the troposphere thermodynamically, a minimum of three coexistent energy fluxes need be considered. To describe convection, we assume

$$J_c(x) \;=\; -\kappa(x)\,\frac{dT}{dx} \;=\; -\kappa(x)\,D\,\psi(x)$$

To describe radiation, in any volume element two independent fluxes are flowing in opposite directions. The gray-gas model presumes a Stefan-Boltzmann distribution of energies with a common absorption distance,

$$[\lambda(x)D+1]J_r^+(x) \;=\; \varphi(x) \qquad \varphi(x) \;=\; \sigma T^4(x)$$
$$[\lambda(x)D-1]J_r^-(x) \;=\; -\varphi(x)$$

J_c decreases with altitude, vanishing at the tropopause (*x=1)*. J_r^+ increases while J_r^- decreases. All may vary in different ways with temperature and altitude. Finding that *T(x)* for which their sum, J_{tot} , is independent of altitude becomes the sought solution.

---

1 https://pdq2021.000webhostapp.com/HBC_Model.pdf

2 https://pdq2021.000webhostapp.com/Thermal_Dissipation_V.pdf

Upon combining these three differential equations,

$$J_{tot}(x) = J_r^+(x) - J_r^-(x) + J_c(x)$$

and

$$-J_{tot} = 2\lambda D\varphi + \kappa D\psi - \lambda D\ \lambda D\ \kappa D\psi$$

The variational solution of this third-order differential equation for *T(x)* is that function minimizing the differences between J_{tot} *(x)* and its mean value.

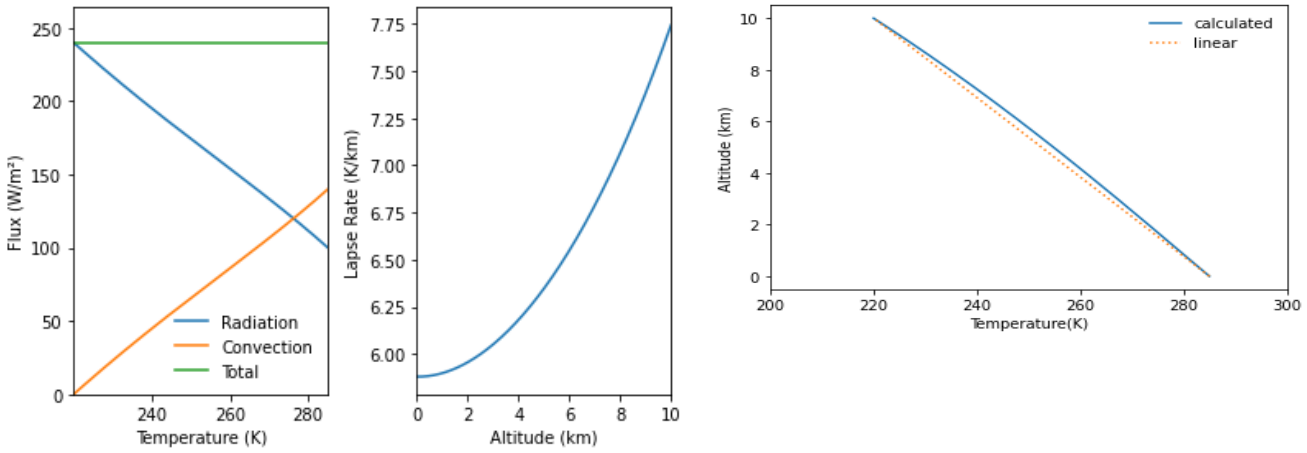
For calculation, we first assign fixed values to T_1, T_2 and the altitude of the troposphere, *e.g. 285K*, 220K and *10 km,* corresponding to a mean lapse rate of *6.5 K/km*. For a linear model let

$\lambda = \lambda(0) * (1 + 2\,x)$

$\kappa = \kappa(0) * (1 - x)$

We expect radiative absorption to vary inversely with the three-fold density decrease in the troposphere and convection to vanish at the tropopause. To complete the calculation, we then seek boundary values for $\lambda(0)$ and $\kappa(0)$ matching desired boundary fluxes, *e.g.* $J_r(0) = 100\ W/m^2/K)$ and $J_r(1) = 240\ W/m^2/K)$. A vintage Windows 7 Dell desktop (one second per trial) yields, by trial and error, $\kappa(0) = 23.8\ W/km/K$ and $\lambda(0) = 1.55$ km , and values

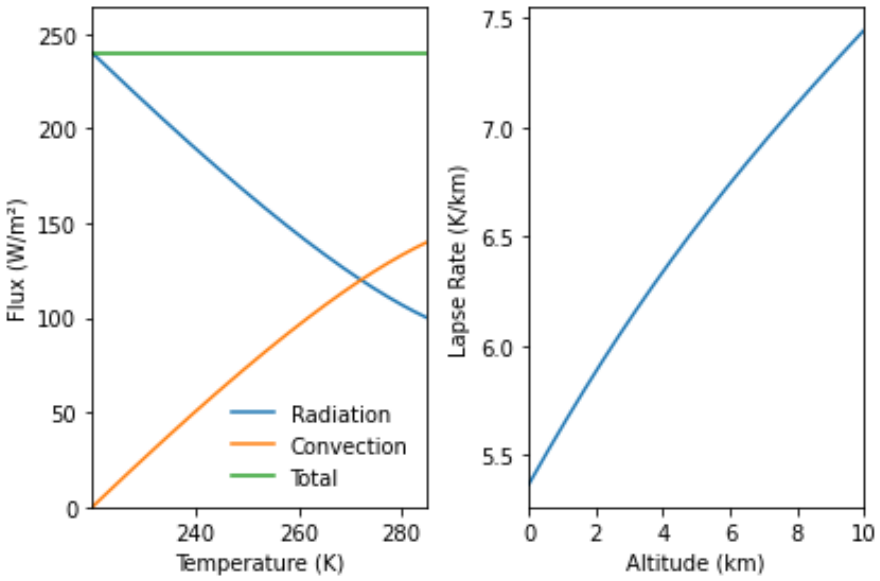
Mean Flux (W/m²): 239.906, Std Dev: 0.000111
Surface Fluxes (W/m²):
Total 239.906
Radiation 99.966
Convection 139.940
T1 Sensitivity: 4.55 (W/m²/K)
T2 Sensitivity: -2.95 (W/m²/K)



Despite the seeming linearity of the third plot, the lapse rate ranges *5.88* to *7.75 K/km*. T1 Sensitivity has been defined as the ratio of a change in total flux to a *0.01K* change in T1, all other parameters fixed. Thus a *3.7 W/m²* flux change (CO_2 doubling?*)* implies a surface temperature increase of *0.81K*.

As an improvement over a linear λ function, we take a known density profile, the U.S. Standard. Atmosphere (USSA) and fit it to a quadratic reciprocal density polynomial, *e.g.* $\lambda = \lambda(0) * (1.0218 + 0.6433\, x + 1.2708\, x^2)$. The result, after revaluing $\lambda(0)$ and $\kappa(0)$,

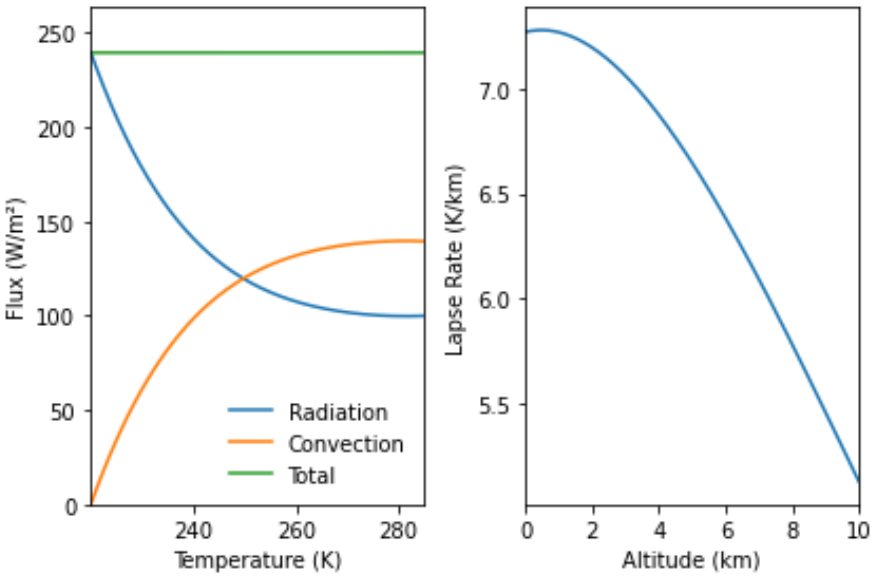
Mean Flux (W/m²): 239.905
Std Dev: 0.000102
Surface Fluxes (W/m²):
Total 239.905
Radiation 99.845
Convection 140.060
T1 Sensitivity: 4.49 (W/m²/K)
T2 Sensitivity: -2.99 (W/m²/K)



For CO_2 doubling, the surface temperature rise is *0.82K* and the lapse rate vs. altitude curvature has changed sign.

To gauge the influence of a nonlinear κ function, suppose there are reasons for reducing radiation and favoring convection at lower altitudes, e.g. $\kappa = \kappa(0) * (1 - x^4)$. After revaluation of $\lambda(0)$ and $\kappa(0)$,

Mean Flux (W/m²): 239.567
Std Dev: 0.003137
Surface Fluxes (W/m²):
Total 239.555
Radiation 99.987
Convection 139.568
T1 Sensitivity: 4.31 (W/m²/K)
T2 Sensitivity: -3.17 (W/m²/K)

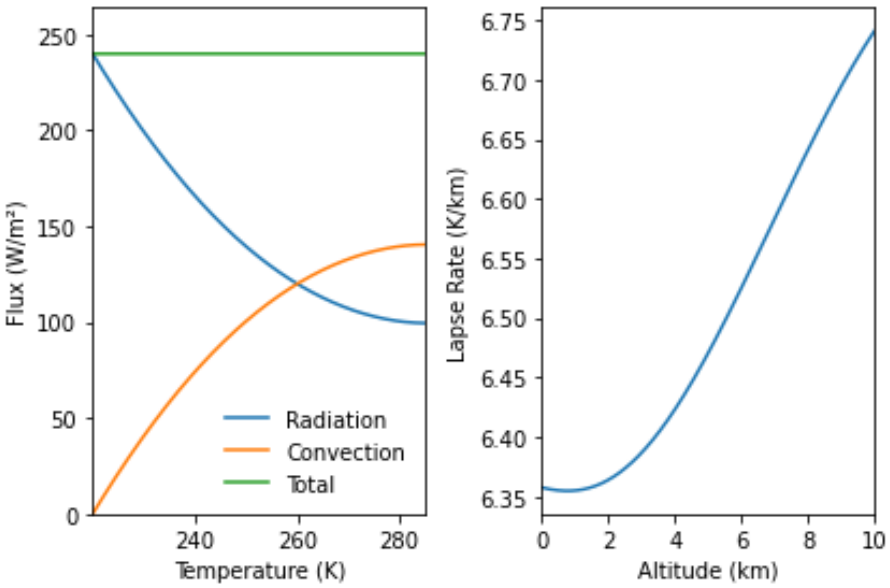


The plots have changed dramatically with the lapse rate now highest at the surface, yet for CO_2 doubling the surface temperature increase is only *0.86K*.

With a less extreme $\kappa = \kappa(0) * (1 - x^2)$,

Mean Flux (W/m²): 240.038
Std Dev: 0.000407
Surface Fluxes (W/m²):
Total 240.036
Radiation 99.513
Convection 140.523
T1 Sensitivity: 4.37 (W/m²/K)
T2 Sensitivity: -3.10 (W/m²/K)

The lapse rate is now nearly constant and, the CO_2 doubling effect *0.85K*.



The most surprising implication of these few thermodynamic model calculations is that thermal sensitivities are virtually independent of how internals of the troposphere are modeled. Greenhouse gas warming can be well-inferred wholly from boundary values for temperature and flux. Lapse rate variations, however, changed dramatically, although their mean value remained fixed at *6.5K/km*.

Currently there seems a quasi-religious belief, particularly prevalent in academic circles, that *1.5K* of global warming constitutes an irreversible 'tipping point', an existential threat for mankind. Salvation demands suffering. To reach this level with the USSA model, let us impose the constraint $\delta T_2 = \delta T_1$. The effective sensitivity then becomes their sum, *4.49 – 2.99 = 1.5 W/m²/K,* and the CO_2 doubling effect *2.47K*. This may be achieved with the *Adiabatic Lapse Rate* presumption that thermal gradients are induced by gravity within an equilibrium system. An equilibrium atmosphere of pure argon would have a lapse rate 18.8 K/km. Not to be found in Climate Science literature is acknowledgment this notion had been originally hypothesized by Kelvin in 1862 as *Convective Equilibrium*. Also not to be found, mention that calculations by Maxwell and Boltzmann revealed molecular velocity distribution functions were not altered by gravitational fields and equilibrium systems are necessarily isothermal. Nor is it to be found that the adiabatic lapse rate itself defines only a necessary condition for the absence of convection. Beyond the pale of academia, several souls have recently dared venture that atmospheres lacking greenhouse gases should be isothermal, but they too have yet to transgress the bounds of Thermodynamics 101.

O tempora, o mores!

P. D. Quondam
10/29/23

Appendix

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 20 09:14:34 2023
For variable names: α β γ δ ε ζ η ϑ ι κ λ μ ν ξ ο π ρ σ τ υ φ χ ψ ω
"""

import numpy as np
import pylab as plt
from scipy.optimize import leastsq
poly = np.polynomial.Polynomial


def Jrf(x, a):
    '''Return  λ(x)*Dφ(x), T(x) and dT/dx '''
    p = poly([0, 1, -1]) * poly(a)   # (x - x^2)*(a + bx + cx^2 +...)
    p = poly([0, 1]) + p             # x + (x-x^2)*(a+bx+..)
    T = T1 + (T2-T1) * p
    dT = T.deriv(1) / height
    f = T**4
    f = f.deriv(1)
    f = poly(λ) * f
    return -2*σ * f(x), T(x), dT(x)


def Jcf(x, a):
    '''Return κ(x)*Dψ(x)  '''
    p = poly([0, 1, -1]) * poly(a)   # (x - x^2)*(a + bx + cx^2 +...)
    p = poly([0, 1]) + p             # x + (x-x^2)*(a+bx+..)
    T = T1 + (T2-T1) * p
    f = T.deriv(1)
    f = f * poly(κ)
    return -f(x)


def Jrcf(x, a):
    '''Return λ(x)*D λ(x)*D κ(x)*Dψ(x)'''
    p = poly([0, 1, -1]) * poly(a)   # (x - x^2)*(a + bx + cx^2 +...)
    p = poly([0, 1]) + p             # x + (x-x^2)*(a+bx+..)
    T = T1 + (T2-T1) * p
    f = poly(κ) * T.deriv(1)
    f = poly(λ) * f.deriv(1)
    f = poly(λ) * f.deriv(1)
    return f(x)


def J(x, a):
    '''Returns the total flux as a function of  parameters in a[]'''
    Jr, _, _ = Jrf(x, a)
    Jc = Jcf(x, a)
    Jrc = Jrcf(x, a)
    return Jr + Jc + Jrc


def Js(x, a):
    '''The variational solution is for a constant value'''
    return np.mean(J(x, a)) * np.ones(len(x))


def residuals(a, J, x):
    return J(x, a) - Js(x, a)


def debug():
    # print (run, λ0, κ0, height)
    print('debug', Jtot[0] - J0)
    return
```

```
x = np.linspace(0, 1, 1001)     # Normalized altitude

# Variable Parameters
T1 = Ta = 285.0         # 285.0         # 285
T2 = Tb = 220.0         # 220.0
height = 10.0           # 10.0                  # altitude of tropopause
delT = .01              # For perturbation
λ0 = 1.63
κ0 = 26.1

# Radiation parameters vary inversely as density for
σ = 5.67e-8                     # Stefan-Boltzmann W/m^2/K
altitude = height * x

#  ................ Model Functions
# Linear
# λ = 1.55 / height * np.array([1, 2.])                     # 1.55 kilometers
# κ = 23.8/ height * np.array([1, -1])                      # 23.8 W/km/K

# US Std Atmosphere
# λ = (1.63 / height) * np.array([1.0218, 0.6433, 1.2708])   # 1.67 kilometers
# κ = (26.1 / height) * np.array([1, -1])                   # 25.0 W/km/K

# Enhanced Surface Convection (4th)
# λ = (1.25 / height) * np.array([1.0218, 0.6433, 1.2708])   # 1.67 kilometers
# κ = (19.2 / height) * np.array([1, 0, 0, 0, -1])          # 25.0 W/km/K

# Enhanced Surface Convection (2th)
# λ = (1.38 / height) * np.array([1.0218, 0.6433, 1.2708])   # 1.38 kilometers
# κ = (22.1 / height) * np.array([1, 0, -1])                # 22.1 W/km/K

# Loop for perturbation calculation of Sensitivity (W/m²/K)
for run in range(0, 6):
    λ = (λ0 / height) * np.array([1.0218, 0.6433, 1.2708])
    κ = (κ0 / height) * np.array([1, -1])
    # debug()
    # Polynomial coefficients for T(x)
    a = np.array([0, 0, 0, 0, 0, 0, 0])     # default initial value

    # Find coefficients making J(x) independent of x
    data = leastsq(residuals, a, args=(J, x))
    a1 = data[0]                            # coefficients of minimization
    y = J(x, a1)
    Jmean, Jstd = np.mean(y),  np.std(y)

    # Find fluxes for calculated temperature function
    Jr, T, dT = Jrf(x, a1)
    Jc = Jcf(x, a1)
    Jrc = Jrcf(x, a1)
    Jtot = Jr + Jc + Jrc
    Jmax = np.max(Jtot)
    if run == 0:
        print('Boundary Temperatures {:5.2f}K, {:5.2f}K'.format(T1, T2))
        print('Mean Flux (W/m²): {:8.3f}, Std Dev: {:8.6f}'.format(Jmean, Jstd))
        print('Surface Fluxes (W/m²):')
        print('   Total      {:8.3f}'.format(Jtot[0]))
        print('   Radiation  {:8.3f}'.format(Jr[0] + Jrc[0]))
        print('   Convection {:8.3f}'.format(Jc[0]))
        # Plot fluxes vs T and lapse rate vs altitude
        fig, axes = plt.subplots(nrows=1, ncols=2)

        # plot 1
        ax = axes[0]
        ax.plot(T, Jr + Jrc, label='Radiation')
        ax.plot(T, Jc, label='Convection')
        ax.plot(T, Jtot, label='Total')
        # ax.plot(T, Jrc, label='Jrc')
        ax.set_xlabel('Temperature (K)')
        ax.set_ylabel('Flux (W/m²)')
        ax.set_xlim(T2, T1)
        ax.legend(frameon=False)
```

```
    ax.set_ylim(0, 1.1*Jmax)

    # plot 2
    ax = axes[1]
    ax.plot(altitude, -dT)
    ax.set_xlabel('Altitude (km)')
    ax.set_ylabel('Lapse Rate (K/km)')
    ax.set_xlim(0, height)
    # ax.set_ylim(5.5, 8)
    fig.tight_layout()
    plt.show()

    # Plot altitude vs. temperature
    plt.plot(T, altitude, label='calculated')
    Y = [altitude[0], altitude[-1]]
    X = [T1, T2]
    plt.plot(X, Y, ls=':', label='linear')
    plt.xlabel('Temperature(K)')
    plt.ylabel('Altitude (km)')
    plt.xlim(Tb, Ta)     # 200, 300)
    plt.legend(frameon=False)
    plt.show()

if run == 0:
    J0 = Jtot[0]
    T1 = Ta + .01

elif run == 1:
    sensitivity = (Jtot[0] - J0) / delT
    print('T1 Sensitivity: {:5.2f} (W/m²/K)'.format(sensitivity))
    T1 = Ta
    T2 = Tb + .01

elif run == 2:
    sensitivity = (Jtot[0] - J0) / delT
    print('T2 Sensitivity: {:5.2f} (W/m²/K)'.format(sensitivity))
    T2 = Tb
    old = height
    height = 1.01 * old

elif run == 3:
    sensitivity = (Jtot[0] - J0) / J0 / (height - old) * old
    print('Flux:Height Correlation {:5.3f})'.format(sensitivity))
    height = old
    old = λ0
    λ0 = 1.01 * old

elif run == 4:
    sensitivity = (Jtot[0] - J0) / J0 / (λ0 - old) * old
    print('Flux:Radiation Correlation {:5.3f})'.format(sensitivity))
    λ0 = old
    old = κ0
    κ0 = 1.01 * old

elif run == 5:  # κ
    sensitivity = (Jtot[0] - J0) / J0 / (κ0 - old) * old
    print('Flux:Convection Correlation {:5.3f})'.format(sensitivity))
    old = κ0 = old

else:
    print('Done')
```