METHOLOGY TO ESTIMATE AVIAN EXPOSURE TO CONCENTRATED SOLAR RADIATION

FOR RIO MESA BIOLOGY RESOURCES APPENDIX BIO1

The principle of thermodynamic equilibrium can be used to calculate the feather surface temperature at any radiant flux density given an ambient air temperature, flight speed of the bird, view factor, and applicable properties of air. This is the method that staff used to estimate surface temperatures resulting from avian exposures to different incident flux densities.

Thus setting Equation 1 equal to Equation 2 and solving for T_S allows estimation of the feather surface temperature at any point in the concentrated flux field.

The equilibrium temperature is dominated by radiant flux input and convective removal of resultant heat to the air passing over the feather surface. In essence the surface temperature will rise until the temperature difference between the air and the surface of the feather ($T_A - T_S$) allows this equilibrium to occur.

Thermodynamic equilibrium equation

$E_F = Q_V$

Where:

E_F - is Radiant heat flux absorbed by the feather surface kW/m^2

Q_V - is heat lost by convection on feather surface kW/m^2

$E_F = E_R A F \varepsilon$ Equation 1

Where:

E_F - is Radiant heat flux absorbed by the feather surface kW/m^2

E_R – is the radiant flux in the concentrated flux field at the birds location

A - is surface area (normalized to $1m^2$)

F – is a dimensionless factor that accounts for shape and orientation of the surface exposed to the Flux

F in staff's analysis assumed the feather is a flat surface and the orientation is adjusted by the Cosine of the angle between the incident flux and the perpendicular to the feather surface.

ε – is (emissivity) a dimensionless factor relating absorption of a surface to that of a black body. Values of emissivity range from 0 to 1. Emissivity of 0 represents no absorption and 1 is 100% absorption.

Convective Heat Transfer

$Q_V = h\ A\ (T_A - T_S)$ Equation 2

Where:

h - is the convective heat transfer coefficient

$h = k_A N_u / L$

A - is surface area (normalized to 1 m^2)

T_A – Ambient temperature

T_S – Temperature on the feather surface

Where:

kA = thermal conductivity of air = .028

N_U = Nussett number = $0.664\ Re^{5} \times Pr^{.33}$

Where:

Pr - the Prandtl number (dimensionless / empirical)

Pr = .705

Staff used Prandtl number for air at 50 ^{o}C

Re - the Reynold's number

Staff used the Reynold's number which is

Re = VL/v

Where:

V - is flight speed of the bird

v – Kinematic viscosity of Air = 1.78 x 10^{-5} m^2/s

L – is the distance from the leading edge of the wing to the trailing edge of the wing

L - used in staff analysis = 6 inches (implies a Reynold's number for laminar flow)

MODEL AND CALCULATIONS TO ESTIMATE AVIAN EXPOSURE TO CONCENTRATED SOLAR RADIATION

FOR RIO MESA BIOLOGY RESOURCES APPENDIX BIO1

Method used in Bird Flight Model

1) Set path conditions
   a) Pick a path through the applicant-provided flux map (provided in **Response to Data Request #159**)
   b) Measure the distances to each of the flux contours across the heliostat field
      i) Assume flux= 0 at edge of field, interpolated elsewhere
   c) Make an interpolation table of distance/flux level.

2) Set environmental and flight conditions
   a) Ambient temperature, Tambient
   b) Flight speed, V
   c) Angle of incidence of flux to feather surface (angle from perpendicular incidence), offVert
      View factor = cosine (offVert)
   d) Wing dimension (leading to trailing edge) L

3) Assume feather's physical properties
   a) Thickness
   b) Optical emissivity = 0.95

c) Optical transmissivity = 0
d) Optical absorption depth = .5 (Assume incident flux is absorbed in first half of thickness, not all at surface)
e) Mass density of keratin
f) Void density (to account for open keratin structure) (assumed 50% of volume)
g) Mass density/unit area of feather
h) Thermal conductivity of keratin
i) Moisture level (delays heating by adding mass)

4) Set initial conditions:
   a) Tsurf = Tambient
   b) Qin = 0
   c) t = 0

5) Start clock (intervals of dt). Repeat the following steps for each clock tick interval, until all way across the heliostat field. Output and graph are stored in viewable files. See Rio Mesa **Appendix BIO1 Figures 1** thorough **4** and **Appendix BIO1 Tables 1** and **2** for examples:
   a) Calculate new time (t) from clock ticks by adding dt (the time interval)
   b) Calculate position along path X = V * t where t = elapsed time, V = flight speed
   c) Calculate Flux Level from position by interpolation between flux contours (from applicant)
   d) Calculate Solar energy received in (Qin) from Flux Level, emissivity, view factor, transmissivity
   e) Calculate hot-side convective energy losses Qv = h * (Tsurf - Tambient)
   f) Calculate hot-side re-radiative losses energy losses Qrad = SBsigma * emissivity * ($Tsurf^4$ - $Tambient^4$)
   g) If backside uncovered, calculate conductive-convective combination losses
      Qcomb = (Tsurf-Tamb) / (thkPlumage * (1-abDepth)/kPlumage + 1/h)
      going through the feather with heat going out the cold-side of the feather; if covered
      Qcomb = 0 <u>(Qcomb set to zero for Rio Mesa)</u>
   h) Calculate energy change during interval as Qnet = Qin – Qv – Qcomb - Qrad
   i) Calculate change in surface temperature during interval dT = Qnet * dt / (CpPlumage * m)
   j) Calculate new surface temperature Tsurf = Tsurf + Dt

BIRD FLIGHT MODEL

FOR RIO MESA BIOLOGY RESOURCES APPENDIX BIO1

Bird Flight Model - Python Open Source Programming Language

http://www.python.org/ - Version 2.7.2.

Printed in mono-spaced font here for readability of computer code.

Selected code extractions

```
# bird plumage characteristics

    Tskin = 41                    #  degC

    transmissivity = 0.0          # of bird feather

    emissivity = 0.95             # of bird feather

    kPlumage = .074               #  W/m-K plumage thermal conductivity

    rhoPlumage = 1.3e3 *.5        #  density in kg/m^3

    thkPlumage =    60e-5 # 2.6e-5   #  0.5e-2    #  meters

    CpPlumage =     1.53e3        #  J/kg-K          ref:

    abDepth = .5                  # fraction of plumage thickness that
                      absorbs the Qin flux

    Tsurf =  Tamb                 # start here for initial temp

    mDryFeather = rhoPlumage * thkPlumage   # feather mass in kg/m^2

    mWater = waterFraction * mDryFeather    # water mass per unit area
                             (kg/m^2 )  adds mass to feathers

    m = mDryFeather + mWater      #water absorbs heat until 100C


    #initialize constants

    viewFactor = math.cos(offVert * math.pi/180.)

    L = L / 39.4        # Convert from inches to meters

    Pr = 0.705          # Prantl number (dimensionless) air MERM App 35.D

    V =  Vmph / 2.237   # convert from mph to meters/sec

    airVis = 1.78e-5    # air kinematic viscosity  MERM App. 35-D
```

```
kAir = .028         # air thermal conductivity W/(m-degK) MERM App 35.D

Qthresh = 4000.     # in watts/m^2

Reynolds = V * L / airVis     # reynolds number

Nu = 0.664 * Reynolds**0.5 * Pr**(.33333333)    #Nusselt number

h = kAir * Nu / L             # convective heat transfer coeff

SBSigma = 5.6704e-8           #  W / (m^2 * K^4) stephan-boltzman constant


# calculations per time interval (dt)

t = i*dt   #new time

d = t*V    #new distance

Qrad = SBSigma * emissivity * ((Tsurf+273)**4 - (Tamb+273)**4)  # in Watts/m^2

Qv =   h * (Tsurf - Tamb)     # 'Front' surface convection in Watts/m^2

SunsIn = qDotIn(d)


 If backSideLossesOn:

    Qin = 1000 * (SunsIn+1) * emissivity * viewFactor * (1 - transmissivity)   # sun is now
                                                                                shining on the topside

     # combined 'backside' conduction + convection in Watts/m^2

    Qcomb = (Tsurf-Tamb) / (thkPlumage * (1-abDepth)/kPlumage + 1/h)
     Tbackside = Qcomb/h + Tamb                  # temperature of back side of feather

     QradBackSide = SBSigma * emissivity * ((Tbackside+273)**4 - (Tamb+273)**4)

     # in Watts/m^2  Rad of energy absorption

     Qnet = Qin - Qv  - Qrad - Qcomb - QradBackSide    # net heat gain during
                                                 clock tick (W/m^2)

Else if not backSideLossesOn:

    Qin = 1000 * SunsIn * emissivity * viewFactor * (1 - transmissivity)   # in Watts

    Qnet = Qin - Qv - Qrad                # net heat gain during clock tick (W/m^2)

    Tbackside = Tsurf

    Qcomb = 0


if not lFeatherIsDry and Tsurf >= 100:      # see is water is gone. if so, remove
                                             its mass

        lFeatherIsDry = True

        m = mDryFeather
```

```
dTemp = Qnet * dt / ( CpPlumage * m * abDepth) #change in temp of feather surface
                                            (front side) during clock tick

    Tsurf += dTemp  #new temp

    doseTotal += Qin * dt
```

Source Code

```
# heat rise of bird surface temperature

# bird_traverse_3e9.py  10/28/2012  Geoff Lesh

# program written in python language, using Matplotlib, numpy extensions (see
imports in _main_)

# compiles and runs with Python ver 2.7.2 (default, Jun 12 2011, 15:08:59)
[MSC v.1500 32 bit (Intel)] on Windows XP

# also uses apple pictureviewer.exe but not needed.

# to run a path, must  uncomment that path paragraph in the pathData()
function. Only one path should be uncommented at a time \

# to prevent confusion.

# most user changes for:  path in pathData()

#                         run settings in userData()

#                         physical constants in setConstants()



def pathData():


    global distVect, nSunsVect, towerLocation, waterFraction, offVert, runID,
emissivity,Tamb, V ,\

            pathID, pathRemarks


    ###flying upward  Note: this path has its own unique scale!

    #scale=300 / 16.7  #meters Real world per cm on map: map data is in same
cm.

    #pathID =  'DAUP'

    #pathRemarks = 'Upward past tower from ground'

    #towerDist = 13.15

    #nSunsVect = (0,5,10,25,50,50,25,10,5,0)

    #distData=   [0,10.8,11.1,11.6,12.3,14,14.4,15.5,15.9,20]  #cm of scale #
```

```
    #pathID = 'Constant 1KW'

    #scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    #pathRemarks = 'Constant 1KW'

    #towerDist = 24.3

    #nSunsVect = (0,1,1,0)

    #distData=   [16.95,17.0, 31.2, 31.25]  #cm of scale #


    #pathID = 'Constant 5KW'

    #pathRemarks = 'Constant 5KW'

    #scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    #towerDist = 24.3

    #nSunsVect = (0,5,5,0)

    #distData=   [16.95,17.0, 31.2, 31.25]  #cm of scale #


    #pathID = 'Constant 8KW'

    #pathRemarks = 'Constant 8KW'

    #scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    #towerDist = 24.3

    #nSunsVect = (0,8,8,0)

    #distData=   [16.95,17.0, 31.2, 31.25]  #cm of scale #


    #pathID = 'Constant 10KW'

    #pathRemarks = 'Constant 10KW'

    #scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    #towerDist = 24.3
```

```
    #nSunsVect = (0,10,10,0)

    #distData=   [16.95,17.0, 31.2, 31.25]  #cm of scale #


    #pathID = 'Constant 25W'

    #pathRemarks = 'Constant 25KW'

    #scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    #towerDist = 24.3

    #nSunsVect = (0,25,25,0)

    #distData=   [16.95,17.0, 31.2, 31.25]  #cm of scale #


    pathID = 'Constant 50KW'

    pathRemarks = 'Constant 50KW'

    scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    towerDist = 24.3

    nSunsVect = (0,50,50,0)

    distData=   [16.95,17.0, 31.2, 31.25]  #cm of scale #


    #pathID = 'Constant 100KW'

    #pathRemarks = 'Constant 100KW'

    #scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    #towerDist = 24.3

    #nSunsVect = (0,100,100,0)

    #distData=   [16.95,17.0, 31.2, 31.25]  #cm of scale #


    #pathID = 'Constant 150KW'

    #pathRemarks = 'Constant 150KW'
```

```
    #scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    #towerDist = 24.3

    #nSunsVect = (0,150,150,0)

    #distData=   [16.95,17.0, 31.2, 31.25]  #cm of scale #


    #pathID = 'AASE'

    #scale=1500./7.7   # meters real world per cm on scale

    #pathRemarks = 'closest pass to tower'

    #towerDist = 21.55

    #distData=   [15.3, 19.4, 20.2, 20.4, 21.2, 21.25, 21.3, 21.65, 21.75,
21.85, 21.95, 22.9, 24.5, 29.5]  #cm of scale # path A1 next to tower

    #nSunsVect = (0,5,10,25,50,100,150,150,100,50,25,10,5,0)
# path A1 next to tower


    #pathID = 'ABNE'

    #pathRemarks = '100 m off tower (tangent)'

    #scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    #towerDist = 20.0

    #nSunsVect = (0,5,10,25,25,10,5,0)
# path ABNE 100 m off tower

    #distData=   [11.7, 17.7, 18.5, 19.0, 21.2, 21.7, 22.5, 31.0]  #cm of
scale #        path ABNE 100 m off tower


    #pathID = 'ACNE' #

    #pathRemarks = '200 m off tower'

    #scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    #towerDist = 20.1

    #nSunsVect = (0,5,10,10,5,0)                                          # path
acNE 200 m off tower
```

```
    #distData=   [12.2,18.2,19.4,19.7,22.7,29.9]  #cm of scale #        path
acNE 200 m off tower



    #pathID = 'ADNE'

    #pathRemarks = '300 m off tower'

    #scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    #towerDist = 21.0

    #nSunsVect = (0,5,10,10,5,0)                                        # path
ADNE 300 m off tower

    #distData=   [13.7,19.3,22.3,23.,23.5,31.0]  #cm of scale #        path
ADNE 300 m off tower



    #pathID = 'AENE'

    #pathRemarks = '400 m off tower'

    #scale=1500./7.7   # meters real world per cm on scale This is general
scale for path not having their own scale

    #towerDist = 24.3

    #nSunsVect = (0,5,5,0)                                         # path AENE
400 m off tower

    #distData=   [17., 22.8, 23.7, 31.2]  #cm of scale #        path AENE 400
m off tower



    if 1:

        distOffSet=distData[0]   # gets subtracted from initial and all
values of distData

        towerLocation= (towerDist - distOffSet) * scale

        checkdata = len(distData)== len(nSunsVect)

        print 'Checkdata:  %s'%checkdata

        if not checkdata:

            print 'distData size: %s'%len(distData)

            print 'nSunsVect size: %s'%len(nSunsVect)
```

```
            raise Exception( 'Data vector lengths do not match.  Quitting.
See output file.'  )

            #sys.exit()

        else:

            for i in zip(distData,nSunsVect):

                print i

            distVect = tuple( scale * (i - distOffSet) for i in distData)  #
in meters

            #distVect = tuple( scale * (i - towerDist) for i in distData)  #
in meters centered at tower



def userData():

    global Tamb, Tskin, dt, emissivity, offVert, L, V, nSteps, waterFraction,
maxDistance,waterFraction, offVert, RunID, emissivity,Tamb, V ,\

           pathID, Vmph, maxTime, transmissivity,backSideLossesOn

    nSteps= 44000

    dt = .01             # seconds, recheck frequency = clock tick

    Tamb = 49.           # degC

    waterFraction = .15  #mass of water



    offVert = 0.         # degrees  angle of incidence  Usually 0 or 71

    L = 6.               # inches wing length front to back

    Vmph = 18.           # mph   of bird flight

    maxDistance = 3000   # meters

    maxTime =  600       # seconds

    backSideLossesOn = False



def setConstants():  # initialize

    global Tamb, dt, density, m, Cp,nSuns,L,Pr, L, Vmph,V, airVis, kAir,Nu,
emissivity, offVert, h, \
```

```
    kPlumage, Tskin, thkPlumage, CpPlumage, viewFactor ,Tsurf,TsurfVect,t,
pathDistVect,IntensityVect, tSecsVect, \

    mDryFeather,timeTo160, timeTo300, timeAbove160, timeAbove300, maxTsurf,
SBSigma,lHit160, lHit300,lFeatherIsDry, doseTotal, \

    doseAbove160,doseAbove300,doseAboveThresh, Qthresh, doseBefore160,
Reynolds, abDepth, transmissivity,backSideLossesOn, \

    maxSunsIn


    #initialize  data vectors

    TsurfVect = []

    pathDistVect = []

    IntensityVect = []

    tSecsVect = []


    #initialize constants

    viewFactor = math.cos(offVert * math.pi/180.)

    L = L / 39.4        # Convert from inches to meters

    Pr = 0.705          # Prantl number (dimensionless)

    V =  Vmph / 2.237   # convert from mph to meters/sec

    airVis = 1.78e-5    # air kinematic viscosity

    kAir = .028         # air thermal conductivity W/(m-degK)

    Qthresh = 4000.     ##in watts/m^2

    Reynolds = V * L / airVis     # reynolds number

    Nu = 0.664 * Reynolds**0.5 * Pr**(.33333333)    #Nusselt number

    h = kAir * Nu / L             # convective heat transfer coeff

    SBSigma = 5.6704e-8           #  W / (m^2 * K^4) stephan-boltzman
constant


    # bird plumage characteristics

    Tskin = 41                     #  degC
```

```
    transmissivity = 0.0           # of bird feather

    emissivity = 0.95              # of bird feather

    kPlumage = .074                #  W/m-K plumage thermal conductivity

    rhoPlumage = 1.3e3 *.5         #  density in kg/m^3

    thkPlumage =    60e-5 # 2.6e-5 #  0.5e-2    #  meters

    CpPlumage =     1.53e3         #  J/kg-K          ref:

    abDepth = .5                   # fraction of plumage thickness that
absorbs the Qin flux

    Tsurf =  Tamb                  # start here for initial temp

    mDryFeather = rhoPlumage * thkPlumage   # feather mass in kg/m^2

    mWater = waterFraction * mDryFeather    # water mass per unit area
(kg/m^2 )  adds mass to feathers

    m = mDryFeather + mWater       #water absorbes heat until 100C




    if True:

        t=0

        maxSunsIn = 0.

        timeTo160 = -99

        timeAbove160 = 0

        timeTo300 = -99

        timeAbove300 = 0

        #maxTsurf = 0

        lHit160 =  False

        lHit300 = False

        lFeatherIsDry = False

        doseTotal = 0

        doseBefore160 = 0

        doseAbove160 = 0
```

```
        doseAbove300 = 0

        doseAboveThresh = 0


def qDotIn(d):

    global i, distVect, nSunsVect

    intensity = np.interp(d,distVect,nSunsVect)

    return intensity


def mainLoop():

    # input data

    global t, Tsurf, nSuns, Tamb, viewFactor, offVert, h, kPlumage, Tskin, m,
V, d, TsurfVect,pathDistVect, IntensityVect, nSteps, \

           thkPlumage,tSecsVect, maxDistance, mDryFeather, maxTime,
timeTo160, timeAbove160, maxTsurf,lHit160, lHit300,\

           timeTo160, timeTo300, timeAbove160,timeAbove300,maxSurfTemp,
lFeatherIsDry, doseTotal, doseAbove160,doseAbove300, \

           doseAboveThresh, Qthresh, doseBefore160, textLines, abDepth,
transmissivity,backSideLossesOn, maxSunsIn


    print "time(s), dist(m),  SolarIn(kW),  Tsurf(degC)    Tbackside    Qin
Qnet,         Qv,     Qcomb,      Qrad"

             #(t, d, SunsIn, Tsurf, Tbackside, Qin, Qnet, Qc, Qcomb, Qrad)


    for i in range(1,nSteps):  # i is clock ticks

        t = i*dt   #new time

        d = t*V  #new distance


        if d > maxDistance:  #don't go beyond edge of solar field

            break

        if t > maxTime:   #don't go beyond maxTime (seconds)
```

```
            break


        SunsIn = qDotIn(d)

        if SunsIn > maxSunsIn:

            maxSunsIn = SunsIn


        if SunsIn == 0.:

            break


        Qrad = SBSigma * emissivity * ((Tsurf+273)**4 - (Tamb+273)**4)  # in
Watts/m^2  Rad of energy absorption


        Qv =   h * (Tsurf - Tamb)                   # 'Front' surface
convection in Watts/m^2




        if backSideLossesOn:

            Qin = 1000 * (SunsIn+1) * emissivity * viewFactor * (1 -
transmissivity)   # in Watts

            Qcomb = (Tsurf-Tamb) / (thkPlumage * (1-abDepth)/kPlumage + 1/h)
# combined 'backside' conduction + convection in Watts/m^2

            Tbackside = Qcomb/h + Tamb                  # temperature of back
side of feather

            QradBackSide = SBSigma * emissivity * ((Tbackside+273)**4 -
(Tamb+273)**4)  # in Watts/m^2  Rad of energy absorption

            Qnet = Qin - Qv  - Qrad - Qcomb - QradBackSide            # net
heat gain during clock tick (W/m^2)


        elif not backSideLossesOn:

            Qin = 1000 * SunsIn * emissivity * viewFactor * (1 -
transmissivity)   # in Watts
```

```
            Qnet = Qin - Qv - Qrad                  # net heat gain during
clock tick (W/m^2)

            Tbackside = Tsurf

            Qcomb = 0



        if not lFeatherIsDry and Tsurf >= 100:      # see is water is gone.
if so, remove its mass

            lFeatherIsDry = True

            m = mDryFeather



        dTemp = Qnet * dt / ( CpPlumage * m * abDepth) #change in temp of
feather surface (front side) during clock tick (assumes all mass
participates)

        Tsurf += dTemp  #new temp



        doseTotal += Qin * dt



        if Tsurf > 160:

            doseAbove160 += Qin * dt

        if Tsurf > 300:

            doseAbove300 += Qin * dt

        if Qin > Qthresh:

            doseAboveThresh += Qin * dt




        tSecsVect.append(t)

        TsurfVect.append(Tsurf)

        pathDistVect.append(d)

        IntensityVect.append(SunsIn)
```

```
        if lHit160 and Tsurf >= 160:
            timeAbove160 +=dt

        if lHit300 and Tsurf >= 300:
            timeAbove300 +=dt

        if Tsurf>=160 and not lHit160:
            lHit160=True
            timeTo160 = t

        if not lHit160:
            doseBefore160 += Qin * dt

        if Tsurf >= 300 and not lHit300:
            lHit300 = True
            timeTo300 = t

        print '%5.0f  , %6.1f,    %6.1f,   %9.1f,   %9.1f,  %9.1f, %9.1f,
%9.1f, %9.1f, %9.1f'\
              %(t, d, SunsIn, Tsurf, Tbackside, Qin, Qnet, Qv, Qcomb, Qrad)

    maxSurfTemp =  max(TsurfVect)
    textLines=[]
    textLines.append(['RunID: %s'%runID])
    textLines.append(['PathID: %s'%pathID])
    textLines.append(['PathRemarks: %s'%pathRemarks])
    textLines.append(['Temp(ambient degC): %4.0f'%Tamb])
    textLines.append(['Speed(mph): %3.0f'%Vmph])
```

```
    textLines.append(['Emissivity: %4.2f'%emissivity])

    textLines.append(['Angle of Incidence (deg): %3.0f'%offVert])

    textLines.append(['View Factor: %4.2f'%viewFactor])

    textLines.append(['Moisture (%%): %3.0f'%(waterFraction * 100)])

    textLines.append(['AbsorbDepthFrac: %3.2f'%(abDepth)])

    textLines.append(['PlumageThk (mils): %8.1f'%(thkPlumage * 39400)])
#converting from meters to mils

    textLines.append(['BackSideLossesOn: %s'%(backSideLossesOn)])
#converting from meters to mils

    textLines.append(['Max Surface Temp(C): %5.0f'% maxSurfTemp])



    print

    for  line in textLines: #

        print line[0]



    print

    print    'Time to  Time above  Time to  Time above (secs)'

    print    '   160C        160C     300C        300C'

    print    ' %5.0f       %5.0f    %5.0f       %5.0f'%(timeTo160,
timeAbove160, timeTo300, timeAbove300)

    print

    print 'Reynolds number:              %9.1f'%(Reynolds)

    print 'Max Suns Intensity:             %7.1f'% maxSunsIn

    print 'Max Surface Temp reached:         %5.0f'% maxSurfTemp

    print 'Flight Speed (ft/min):          %7.1f (%7.1f mph)'%
(Vmph*5280/60., Vmph)

    print 'Total flight time (secs):       %7.0f'%(t)

    print 'Dose_total (kW-secs/m^2):       %7.1f'% (doseTotal/60000.*60)

    print 'DoseBefore160 (kW-secs/m^2):    %7.1f'% (doseBefore160/60000.*60)

    print 'DoseAbove160 (kW-secs/m^2):     %7.1f'% (doseAbove160/60000.*60)
```

```
    print 'DoseAbove300 (kW-secs/m^2):     %7.1f'% (doseAbove300/60000.*60)

    print 'DoseAboveThresh (kW-secs/m^2):  %7.1f'%
(doseAboveThresh/60000.*60)



def makePlot():

    global pathDistVect, IntensityVect, TsurfVect, tSecsVect, towerLocation,
distVect,waterFraction, offVert, runID,emissivity,Tamb, V ,\

           pathID,Vmph,pathRemarks, viewFactor, timeTo160, timeAbove160,
timeTo300, timeAbove300,maxSurfTemp, fname, textLines




    newIntensity = [a for a in IntensityVect]

    pathDistVectMod = [a- towerLocation for a in  pathDistVect]

    distVectMod = [a- towerLocation for a in  distVect]  # these are the
markers for the field map countour measurements

    maxIntensity = max(newIntensity)

    plt = matplotlib.pyplot


    host = host_subplot(111, axes_class=AA.Axes)

    plt.subplots_adjust(right=0.75)

    plt.subplots_adjust(bottom= 0.180)


    par1 = host.twinx()

    par2 = host.twiny()


    offset = 60

    new_fixed_axis = par2.get_grid_helper().new_fixed_axis

    par2.axis["bottom"] = new_fixed_axis(loc="bottom",

                                         axes=par2,
```

```
                                        offset=(0, -35))


    par2.axis["bottom"].toggle(all =  True)

    par2.axis["top"].toggle(all =  False)



    host.set_ylim(0, maxSurfTemp*1.05)

    par1.set_ylim(0,1.05*maxIntensity)

    host.set_xlabel("distance (m)")

    host.set_ylabel("Surface Temp (degC) (dashed line)")

    par2.grid(True)

    par1.set_ylabel("Field Intensity (kw = #Suns) (solid line)")

    par2.set_xlabel("time(seconds)")


    p1, = host.plot(pathDistVectMod, TsurfVect,'r--')

    p2, = par1.plot(pathDistVectMod,newIntensity)# , label="kW (= Suns)")

    p3, = par2.plot(tSecsVect, TsurfVect, alpha=0)# ,label="time")

    p4, = par1.plot(distVectMod, nSunsVect, 's', markersize=4,
markerfacecolor='blue',markeredgecolor='blue')

    if timeTo160 > 0:

        jjl=host.axhspan(160,160,0.0,0.75,color='r', linewidth=.5)

        jj2=par2.text(tSecsVect[int(len(tSecsVect)*.83)],156,'%4.0f secs to
reach 160 degC'%timeTo160,color='r', horizontalalignment='left',

                verticalalignment='top', fontsize = 'x-small')#,transform =
host.transAxes)

        jj2=par2.text(tSecsVect[int(len(tSecsVect)*.83)],164,'%4.0f secs
above'%timeAbove160,color='r', horizontalalignment='left',

                verticalalignment='bottom', fontsize = 'x-small')#,transform
= host.transAxes)
```

```
    if timeTo300 > 0:

        Tval=300

        jjl=host.axhspan(Tval,Tval,0.0,0.75,color='r', linewidth=.5)

        jj2=par2.text(tSecsVect[int(len(tSecsVect)*.83)],Tval-4,'%4.0f secs
to reach 300degC'%timeTo300,color='r', \

                      horizontalalignment='left',

            verticalalignment='top', fontsize = 'x-small')#,transform =
host.transAxes)

        jj2=par2.text(tSecsVect[int(len(tSecsVect)*.83)],Tval+4,'%4.0f secs
above'%timeAbove300,color='r', horizontalalignment='left',

            verticalalignment='bottom', fontsize = 'x-small')#,transform =
host.transAxes)



    #par1.set_ylim(0, 4)

    #par2.set_ylim(1, 65)




    host.axis["left"].label.set_color(p1.get_color())

    par1.axis["right"].label.set_color(p2.get_color())



    par2Span=(host.axis()[1]-host.axis()[0])/V

    par2.set_xlim(0,par2Span)



    plt.title(r'$\mathrm{Feather\ Surface\ Temperature\ along\ Flight\ Path\
}$')



    for  line in enumerate(textLines): #

        ##incr x, incr y

        host.text(0.01, .98-line[0]*.036,line[1][0], \
```

```
            horizontalalignment='left',
            verticalalignment='top',
            fontsize = 9,
            transform = host.transAxes)


    fullFname=str('c:\\mypython\\birds\\%s.png'%fname)
    myStr='saved to '+ fullFname
    print myStr
    plt.savefig('c:\\mypython\\birds\\%s.png'%fname)
    appfile= "c:\\program files\\quicktime\\pictureviewer.exe "
    subprocess.Popen([appfile, fullFname] )  # to view the plot
    #plt.show()  #Tk causes prolems? after second plot won't close!

if __name__ == "__main__":
    try:
        import math
        import sys
        import datetime
        import math
        import numpy as np
        import matplotlib
        import matplotlib.pylab
        from mpl_toolkits.axes_grid1 import host_subplot
        import mpl_toolkits.axisartist as AA
        from datetime import datetime
        import subprocess
        runID = '%20s'%str(datetime.now())[:19] #'Dummy' #fixme
```

```
fname=runID.replace(':','')
fname2=fname.replace('.','')
fname='Bird'+fname2

textFileName=str('c:\\mypython\\birds\\%s.txt'%fname)
print 'output is being redirected to : %s'%textFileName
sys.stdout = open(textFileName,'w')

print datetime.now().ctime()
print 'This text file: %s'%textFileName
print "program: bird_traverse_3d.py version 0.5"
print 'program: %s'%(sys.argv[0])

userData()
setConstants()
pathData()
mainLoop()
sys.stdout = sys.__stdout__
print 'Time(s)  Dist(m)   Tsurf(C)     Intensity(suns)'
for a in zip(tSecsVect,pathDistVect,TsurfVect, IntensityVect):
    print '%5.0f  , %5.0f  , %6.1f  ,   %5.1f'%a # (a[0],a[1],a[2])

print

for  line in textLines: #
    print line[0]

print
```

```
        print    'Time to  Time above  Time to  Time above (secs)'

        print    '   160C        160C     300C        300C'

        print ' %5.0f       %5.0f    %5.0f       %5.0f'%(timeTo160,
timeAbove160, timeTo300, timeAbove300)

        print

        print 'Max Surface Temp(C):             %5.0f'% maxSurfTemp

        print 'Reynolds number:               %9.1f'%(Reynolds)

        print 'Max Suns Intensity:             %7.1f'% maxSunsIn

        print 'Flight Speed (ft/min):           %7.1f (%3.1f mph)'%
(Vmph*5280/60., Vmph)

        print 'Total flight time (secs):     %7.0f'%(t)

        print 'Dose_total (kW-secs/m^2):       %7.1f'% (doseTotal/60000.*60)

        print 'doseBefore160 (kW-secs/m^2):    %7.1f'%
(doseBefore160/60000.*60)

        print 'DoseAbove160 (kW-secs/m^2):     %7.1f'%
(doseAbove160/60000.*60)

        print 'DoseAbove300 (kW-secs/m^2):     %7.1f'%
(doseAbove300/60000.*60)

        print 'DoseAboveThresh (kW-secs/m^2):  %7.1f'%
(doseAboveThresh/60000.*60)

        print 'BackSideLossesOn: %s'%(backSideLossesOn)



        makePlot()

        print 'This text file: %s'%textFileName



        print 'program: sys.argv[0] = %s'%sys.argv[0]



    finally:

        sys.stdout = sys.__stdout__  #restore stdout back to normal

        print "done."
```