RadioCalcs

September 6, 2019

1 Radio Calculation Notebook

What it says on the tin. Used to calculate values concerning radio placement of sector antennas.

1.1 Variables

There are many variables that can be calculated for, and a constant of *Omega 1*.
Altitude, (alt) - altitude of the node
Target Alt, (tgt) - altitude of the radio beam at a given distance
Top Alt, (top) - theoretical top of the beam height at a given distance
Bottom Alt, (bot) - theoretical bottom of the beam height at a given distance
Omega 1, (O1) - beam angle of the sector antenna, 8 degrees (vertical); this is a *constant*
Omega 2, (O2) - vertical tilt of the sector itself; if this is >4 degrees the beam will never intersect the ground
Distance, (dist) - horizontal distance from the node to the area needing coverage

1.2 Assumptions

- Line of Sight (**LOS**)
  - Critical; atmospheric refraction isn't enough to overcome the curvature of the Earth
- All of this is theoretical (currently) and highly experimental
  - This is effectively a LOS calculator
  - This doesn't take into account interferance, Fresnel zone intrusions, trasmit power over the distance
- The user is capable of figuring out if given values will work
  - LOS is important but transmit power is critical
  - Flight Example
    - Lt Col Riley flew with a 4200 radio in a T-6; we aimed a 4400-sector at him
      - We could send signals out at ~55 miles
      - We couldn't really hear back from his radio
      - This is because the 4200 with 2dB antennas was insufficient to send back a good signal at that range
      - We also had issues with aiming the antennas properly

- The values of these calculations are not bounded by physics - they are simple geometry problems
  - There is no propagation limitation, there is no distance maximum, there are no viewshed calculations built in
  - The refraction of the atmosphere is not baked in
  - The Earth's curvature is not built in
- This tool is to be used to find (basically) the distances and angles needed, with no limitation
  - If you want a *top* of 57k feet, and a *bot* of 0 feet, the distance is going to be extremely large, and may greatly exceed the distances the radios can actually cover

VariablesVisual.JPG

[1]:
```
from math import tan
from math import atan
from math import radians
from math import degrees
```

[2]:
```
# solve for needed radio Altitude
# alt = top-(dist)tan(O1/2)-(dist)tan(O2)  [top, dist, O1, O2]
#     = bot+(dist)tab(O1/2)-(dist)tan(O2)  [bot, dist, O1, O2]
#     = tgt-(dist)tan(O2)                  [tgt, dist, O2]

def find_alt(top=None, tgt=None, bot=None, dist=None, O1=8, O2=None):
    alt=0
    O1, O2=radians(O1), radians(O2)
    """
    Given specific combos of parameters, returns the needed altitude
        [top, dist, O1, O2] or
        [bot, dist, O1, O2] or
        [tgt, dist, O2]
    """
    if [top, bot, tgt, dist, O1, O2].count(None)==0:
        return "Leave a param blank to solve for."
    if [top, dist, O1, O2].count(None)==0:
        #eq 1
        alt=top-dist*tan(O2+O1/2)
    elif [bot, dist, O1, O2].count(None)==0:
        #eq 2
        alt=bot-dist*tan(O2-O1/2)
    elif [tgt, dist, O2].count(None)==0:
        #eq 3
        alt=tgt-dist*tan(O2)
    else:
        return "Variable combo not recognized"
```

```
    return max(alt, 0)
```

[3]:
```
# solve for needed Target Alt
# tgt = alt+(dist)tan(O2)                    [alt, dist, O2]
#     = top-(dist)*[tan(O2+O1/2)-tan(O2)]    [top, dist, O1, O2]
#     = bot-(dist)*[tan(O2-O1/2)-tan(O2)]    [bot, dist, O1, O2]

def find_tgt(alt=None, top=None, bot=None, dist=None, O1=8, O2=None):
    tgt=0
    O1, O2=radians(O1), radians(O2)
    """
    Given specific combos of parameters, returns the target altitude
        [alt, dist, O2] or
        [top, dist, O1, O2] or
        [bot, dist, O1, O2]
    """
    if [alt, top, bot, dist, O1, O2].count(None)==0:
        return "Leave a param blank to solve for."
    if [alt, dist, O2].count(None)==0:
        #eq 1
        tgt = alt+dist*tan(O2)
    elif [top, dist, O1, O2].count(None)==0:
        #eq 2
        tgt = top-(dist)*(tan(O2+O1/2)-tan(O2))
    elif [bot, dist, O1, O2].count(None)==0:
        #eq 3
        tgt = bot-(dist)*(tan(O2-O1/2)-tan(O2))
    else:
        return "Variable combo not recognized"
    return max(tgt, 0)
```

[4]:
```
## TODO: Add equation based on "tgt"

# solve for expected Top Alt
# top = alt+(dist)*tan(O2+O1/2)                  [alt, dist, O1, O2]
#     = bot-(dist)*[tan(O2-O1/2)-tan(O2+O1/2)]   [bot, dist, O1, O2]

def find_top(alt=None, tgt=None, bot=None, dist=None, O1=8, O2=None):
    top=0
    O1, O2=radians(O1), radians(O2)
    """
    Given specific combos of parameters, returns the target altitude
        [alt, dist, O1, O2] or
        [bot, dist, O1, O2]
    """
    if [alt, bot, dist, O1, O2].count(None)==0:
        return "Leave a param blank to solve for."
    if [alt, dist, O1, O2].count(None)==0:
```

```
        #eq 1
        top = alt+(dist)*tan(O2+O1/2)
    elif [bot, dist, O1, O2].count(None)==0:
        #eq 2
        top = bot-(dist)*(tan(O2-O1/2)-tan(O2+O1/2))
    else:
        return "Variable combo not recognized"
    return max(top, 0)
```

[5]:
```
## TODO: Add equation based on "tgt"

# solve for expected Bottom Alt
# bot = alt+(dist)*tan(O2-O1/2)                    [alt, dist, O1, O2]
#     = top-(dist)*[tan(O2+O1/2)-tan(O2-O1/2)]   [top, dist, O1, O2]

def find_bot(alt=None, top=None, tgt=None, dist=None, O1=8, O2=None):
    bot=0
    O1, O2=radians(O1), radians(O2)
    """
    Given specific combos of parameters, returns the target altitude
        [alt, dist, O1, O2] or
        [top, dist, O1, O2]
    """
    if [alt, top, tgt, dist, O1, O2].count(None)==0:
        return "Leave a param blank to solve for."
    if [alt, dist, O1, O2].count(None)==0:
        #eq 1
        bot = alt+(dist)*tan(O2-O1/2)
    elif [top, dist, O1, O2].count(None)==0:
        #eq 2
        bot = top-(dist)*(tan(O2+O1/2)-tan(O2-O1/2))
    else:
        return "Variable combo not recognized"
    return max(bot, 0)
```

[6]:
```
# solve for needed Omega 2
# O2 = tan^(-1)((tgt-alt)/dist)              [alt, tgt, dist]
#    = tan^(-1)((top-alt)/dist)-O1/2         [alt, top, dist, O1]
#    = tan^(-1)((bot-alt)/dist)+O1/2         [alt, bot, dist, O1]

def find_O2(alt=None, top=None, tgt=None, bot=None, dist=None, O1=8):
    O2=0
    O1=radians(O1)
    """
    Given specific combos of parameters, returns the target altitude
        [alt, tgt, dist] or
        [alt, top, dist, O1] or
        [alt, bot, dist, O1]
```

```
    """
    if [alt, tgt, top, bot, dist, O1].count(None)==0:
        return "Leave a param blank to solve for."
    if [tgt, alt, dist].count(None)==0:
        #eq 1
        O2 = atan((tgt-alt)/dist)
    elif [top, alt, dist, O1].count(None)==0:
        #eq 2
        O2 = atan((top-alt)/dist)-O1/2
    elif [bot, alt, dist, O1].count(None)==0:
        #eq 3
        O2 = atan((bot-alt)/dist)+O1/2
    else:
        return "Variable combo not recognized"
    return degrees(O2)
```

[7]:
```
# solve for needed Distance
# dist = (bot-alt)/tan(O2-O1/2)   [alt, bot, O1, O2]
#      = (top-alt)/tan(O2-O1/2)   [alt, top, O1, O2]
#      = (tgt-alt)/tan(O2)        [alt, tgt, O2]

def find_dist(alt=None, top=None, tgt=None, bot=None, O1=8, O2=None):
    dist=0
    O1, O2=radians(O1), radians(O2)
    """
    Given specific combos of parameters, returns the needed distance
        [alt, bot, O1, O2] or
        [alt, top, O1, O2] or
        [alt, tgt, O2]
    """
    if [alt, tgt, top, bot, O1, O2].count(None)==0:
        return "Leave a param blank to solve for."
    if [alt, bot, O1, O2].count(None)==0:
        dist = (bot-alt)/tan(O2-O1/2)
    elif [alt, top, O1, O2].count(None)==0:
        dist = (top-alt)/tan(O2+O1/2)
    elif [alt, tgt, O2].count(None)==0:
        dist = (tgt-alt)/tan(O2)
    else:
        return "Variable combo not recognized"
    return dist
```