In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from IPython.display import display, Markdown

skip_plots = False
```

Define the bounds of our data:

- Weights
- Indexes corresponding to weights for easier lookup
- Altitudes
- Indexes corresponding to altitudes for easier lookup
- Number of points in the linspaces used for fitting and plotting

In [ ]:
```
weights         = [140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280,
weights_index   = [  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,  14,
alts            = [250, 270, 290, 310, 330, 350, 370, 390, 410, 430]
alts_index      = [  0,   1,   2,   3,   4,   5,   6,   7,   8,   9]
points_for_fit  = 1000
```

Transform functions for altitudes and weights to their respective indices and back.

In [ ]:
```
def weight2index(weight: int) -> int:
  """Return the weight as a weight index."""
  return int((weight - 140) / 10)

def index2weight(index: int) -> int:
  """Return the weight index as a weight."""
  return 140 + index * 10

def alt2index(alt: int) -> int:
  """Return the altitude as an altitude index."""
  return int((alt - 250) / 20)

def index2alt(index: int) -> int:
  """Return the altitude index as an altitude."""
  return 250 + index * 20
```

Very basic IAS to TAS conversion based ona a speed gain of 2% per 1000ft.

In [ ]:
```
def ias2tas(ias: int, alt: int) -> float:
  """Return TAS from IAS and altitude."""
  return ias + (alt/10.0 * (ias * 0.02))
```

Parsing function to read in the CRZ PERF data files and construct a key value list for all known tables.

See `Format description.md` for a description of the file format.

In [ ]:
```
matrix = {}

def parseFile(file_path: str) -> dict:
  """
  Parse file and return as matrix.

  Matrix has altitude keys and (Fuel Flow, IAS, mach) as values
  """
  matrix = {}
  file = open(file_path, 'r')
  lines = file.readlines()
```

```
  count = 0
  for line in lines:
    count += 1
    if count == 1:
      continue
    fields = line.split(";")
    alt = int(fields[0]) / 100
    ff = [int(item.strip()) for item in fields[1].split(",")]
    ias = [int(item.strip()) for item in fields[2].split(",")]
    mach = [float(item.strip()) for item in fields[3].split(",")]

    inner = {}
    for i in range(len(ff)):
      _ias = ias[i] if len(ias) > 1 else ias[0]
      _mach = mach[i] if len(mach) > 1 else mach[0]
      inner[weights[i]] = (ff[i], _ias, _mach)
    matrix[alt] = inner
  return matrix

matrix["LRC"] = parseFile("LRC.data")
matrix[82] = parseFile(".82.data")
matrix[83] = parseFile(".83.data")
matrix[84] = parseFile(".84.data")
```

Basic parametrized functions used for the fitting.

In [ ]:
```
def func_quad(x: np.ndarray, a: float, b: float, c: float) -> np.ndarray:
  """Return f(x) = ax^2 + bx + c."""
  return a * x**2 + b * x + c

def func_root(x: np.ndarray, a: float, b: float, c: float) -> np.ndarray:
  """Return f(x) = a*sqrt(x) + bx + c."""
  return a * np.sqrt(x) + b * x + c
```

Quick lookup of $\mathrm{LRC}mach$ from our matrix.

In [ ]:
```
def get_lrc_mach(weight: int, alt: int) -> float:
  """Return LRC mach at given altitude and weight."""
  return matrix["LRC"][alt][weight][2]
```

Generate "base" lists.

Base lists are extracted from the matrix for a given weight, altitude, and expected $\mathrm{MRC}mach$.

Returned values are base lists for mach, mileage, colour (for the plot). Additional returned value is the $\mathrm{MRC}mileage$ and $\mathrm{mMRC}mach$.

$\mathrm{MRC}mileage$ is calculated as the value required to get $\mathrm{LRC}mileage$ if $\mathrm{LRC}mileage$ is a 1% drop from $\mathrm{MRC}mileage$.

In [ ]:
```
def generate_base_lists(weight: int, alt: int, mrc_mach: float) -> tuple[np.ndarray, np.ndarr
  """
  For a given weight, altitude, and MRC mach, return tuple containing
  - mach_base: [MRC mach, LRC mach, .82, .83, .84]
  - mileage_base: [MRC mileage, LRC mileage, .82 mileage, .83 mileage, .84 mileage]
  - col: Colour list for plotting, green is MRC; orange is LRC; .82, .83, .84 are blue; unknou
  - mrc_mileage: Calculated MRC mileage
  - mrc_mach: Pass through of MRC mach
  """
  raw = []
  col = []
  try:
```

```
    for spd in range(82, 85):
      pt = matrix[spd][alt][weight]
      raw.append((pt[0], ias2tas(pt[1], alt), pt[2]))
  except KeyError:
    pass

  lrc = matrix["LRC"][alt][weight]
  raw.append((lrc[0], ias2tas(lrc[1], alt), lrc[2], "LRC"))
  raw.sort(key=lambda i: i[2])

  spd_base = np.array([x[1] for x in raw])
  ff_base = np.array([x[0] * 3 for x in raw])
  mileage_base = spd_base / ff_base * 1000
  index = next((i for i, item in enumerate(raw) if len(item) == 4 and item[3] == 'LRC'), -10)
  lrc_mileage = mileage_base[index]
  mrc_mileage = lrc_mileage * 100.0/99.0

  raw.append((-1, -1, mrc_mach, "MRC"))
  raw.sort(key=lambda i: i[2])
  for i in range(len(raw)):
     if len(raw[i]) == 4 and raw[i][3] == "MRC":
        mileage_base = np.insert(mileage_base, i, mrc_mileage)
        break

  for x in raw:
    if len(x) < 4:
      col.append("blue")
    elif x[3] == "MRC":
      col.append("green")
    elif x[3] == "LRC":
      col.append("orange")
    else:
      col.append("red")

  mach_base = np.array([x[2] for x in raw])
  return mach_base, mileage_base, col, mrc_mileage, mrc_mach
```

Iteratively generate potential $\mathrm{MRC}_{\mathrm{mach}}$ numbers for a given weight and altitude.

Use $\mathrm{MRC}_{\mathrm{mach}}$, $\mathrm{LRC}_{\mathrm{mach}}$, .82, .83, and .84 as the x and mileage as the y to fit a quadratic function.

Returned values are the $\mathrm{MRC}_{\mathrm{mach}}$ value that has the lowest absolute error to the `max()` of the fitted function, the x axis linspace, and the corresponding fitted function.

Optionally plot all generate curves.

Optionally print altitude, weight, $\mathrm{MRC}_{\mathrm{mach}}$, calculated $\mathrm{MRC}_{\mathrm{mileage}}$, `max()` $\mathrm{MRC}_{\mathrm{mileage}}$, and the absolute difference of the latter two.

In [ ]:
```
def find_best_mrc_mach(weight: int, alt: int, show_intermediate_graphs: bool = False, debug_o
  """
  For a given weight and altitude, return tuple containing
  - best_mrc_mach: MRC mach closest to the max() of the mach/mileage curve
  - x: x axis values for the mach/mileage curve
  - best_func: mach/mileage curve

  Optional parameters:
  - show_intermediate_graphs: Default False. Used to plot all curves.
  - debug_output: Default False. Used for debug
  """
  lrc_mach = get_lrc_mach(weight, alt)
  rng = np.linspace(lrc_mach - 0.1, lrc_mach, 100, False)
  best_mrc_mach = 0
  best_func = 0
```

```
  last_mrc_mileage_error = 10000

  for mrc_mach in rng:
    mach, mileage, colour, mrc_mileage, _ = generate_base_lists(weight, alt, mrc_mach)
    x = np.linspace(mrc_mach - 0.1, 0.95, points_for_fit)

    popt, _ = sp.optimize.curve_fit(func_quad, mach, mileage, maxfev=100000)
    mileage_mach_func = func_quad(x, *popt)

    if debug_output:
      print(f"{alt=}, {weight=}, {mrc_mach=}", mrc_mileage, max(mileage_mach_func), abs(mrc_m

    if abs(mrc_mileage - max(mileage_mach_func)) < last_mrc_mileage_error:
      last_mrc_mileage_error = abs(mrc_mileage - max(mileage_mach_func))
      best_mrc_mach = round(mrc_mach, 3)
      best_func = mileage_mach_func
    else:
      return best_mrc_mach, x, best_func

    if show_intermediate_graphs and not skip_plots:
      plt.figure(figsize=(20,10))
      plt.title(f"mach/Mileage for {weight=} and {mrc_mach=}")
      plt.plot(x, mileage_mach_func)
      plt.scatter(mach, mileage, c=colour)
      plt.show()
```

Generate all curves for all weight and altitude combinations.

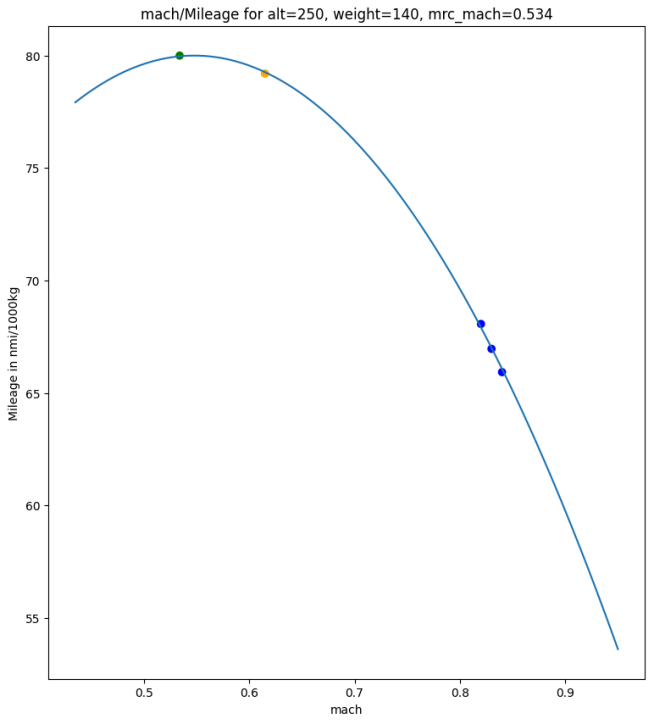
Use the mach/mileage curves to generate CI/mach curves for all weight and altitude combinations by fitting a root function using 0, 200, and 999 as the x and $\mathrm{MRCmach}$, $\mathrm{LRCmach}$, and 0.85 as the corresponding y.

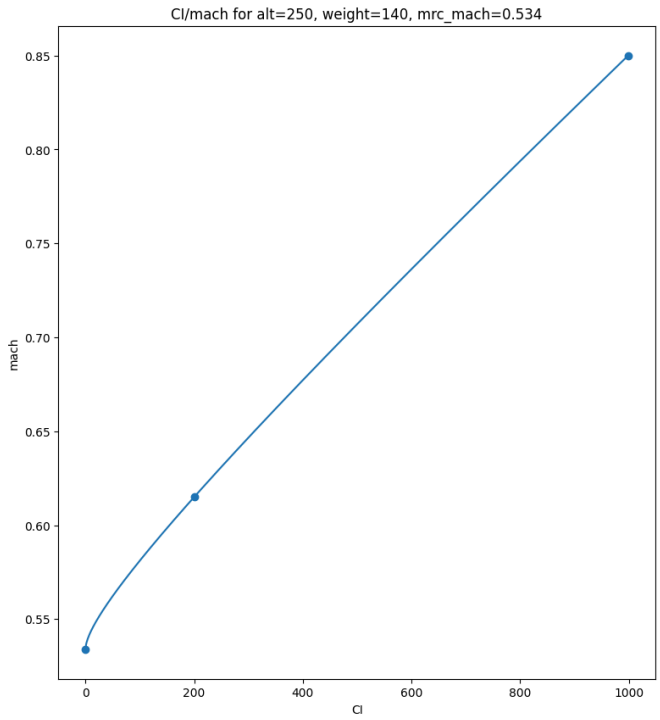
In [ ]:
```
curves = []
for alt in alts:
  _curves = []
  for weight in weights:
    try:
      lrc_mach = get_lrc_mach(weight, alt)
      best_mrc_mach, x, mileage_mach_func = find_best_mrc_mach(weight, alt)
      display(Markdown(f"**Found: {best_mrc_mach} MRC and {lrc_mach} LRC**"))

      mach, mileage, colour, mrc_mileage, mrc_mach = generate_base_lists(weight, alt, best_mr

      if not skip_plots:
        plt.figure(figsize=(20,10))

        plt.subplot(1, 2, 1)
        plt.title(f"mach/Mileage for {alt=}, {weight=}, {mrc_mach=}")
        plt.xlabel("mach")
        plt.ylabel("Mileage in nmi/1000kg")
        plt.plot(x, mileage_mach_func)
        plt.scatter(mach, mileage, c=colour)

      ci_mach_base = [best_mrc_mach, lrc_mach, 0.85]
      ci_base = [0, 200, 999]
      x = np.linspace(0, 999, points_for_fit)

      popt, _ = sp.optimize.curve_fit(func_root, ci_base, ci_mach_base, maxfev=10000)
      ci_mach_func = func_root(x, *popt)

      if not skip_plots:
        plt.subplot(1, 2, 2)
        plt.title(f"CI/mach for {alt=}, {weight=}, {mrc_mach=}")
        plt.xlabel("CI")
```

```
        plt.ylabel("mach")
        plt.scatter(ci_base, ci_mach_base)
        plt.plot(x, ci_mach_func)
        plt.show()

      display(Markdown(f"**Sanity check: {ci_mach_func[0]} MRC and {round(ci_mach_func[int(po

      _curves.append(ci_mach_func)
    except KeyError:
      pass
  curves.append(_curves)
```

Found: 0.534 MRC and 0.615 LRC

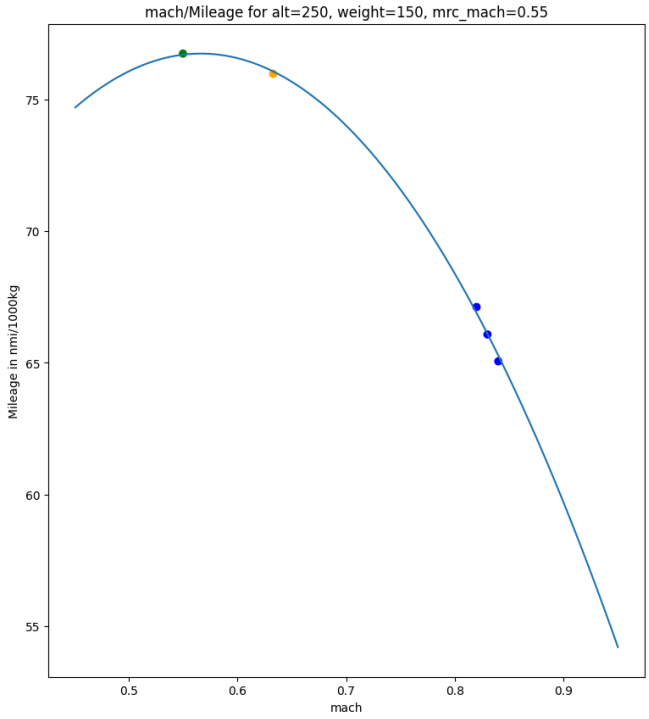
```
C:\Users\llego\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\Local
Cache\local-packages\Python311\site-packages\scipy\optimize\_minpack_py.py:1010: OptimizeWarni
ng: Covariance of the parameters could not be estimated
  warnings.warn('Covariance of the parameters could not be estimated',
```

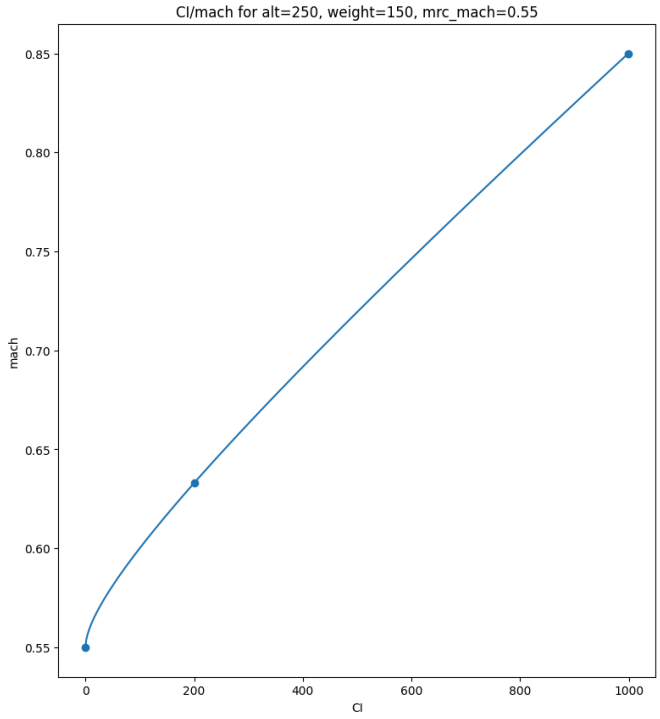




Sanity check: 0.534 MRC and 0.615 LRC

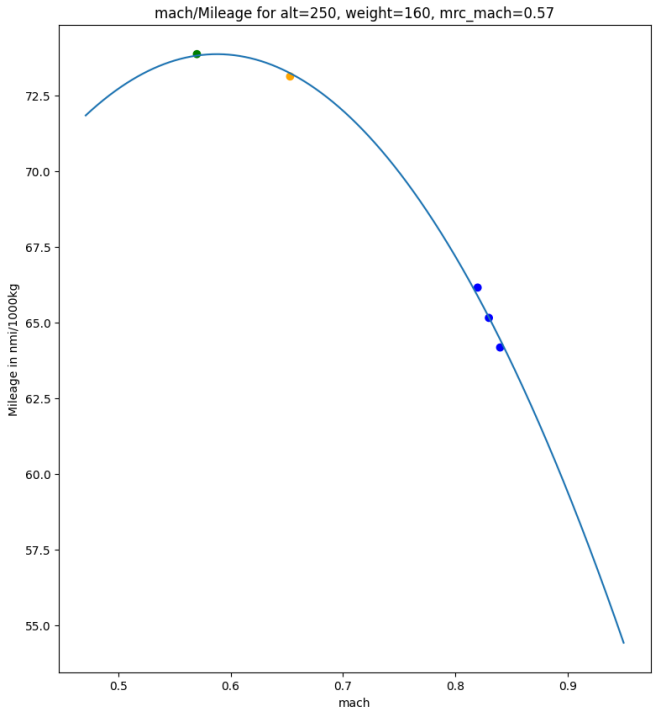
Found: 0.55 MRC and 0.633 LRC

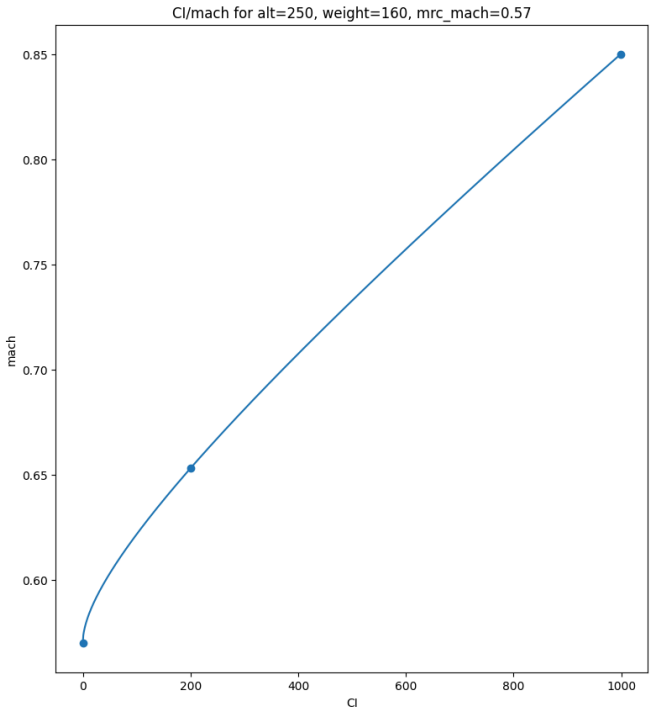




Sanity check: 0.55 MRC and 0.633 LRC

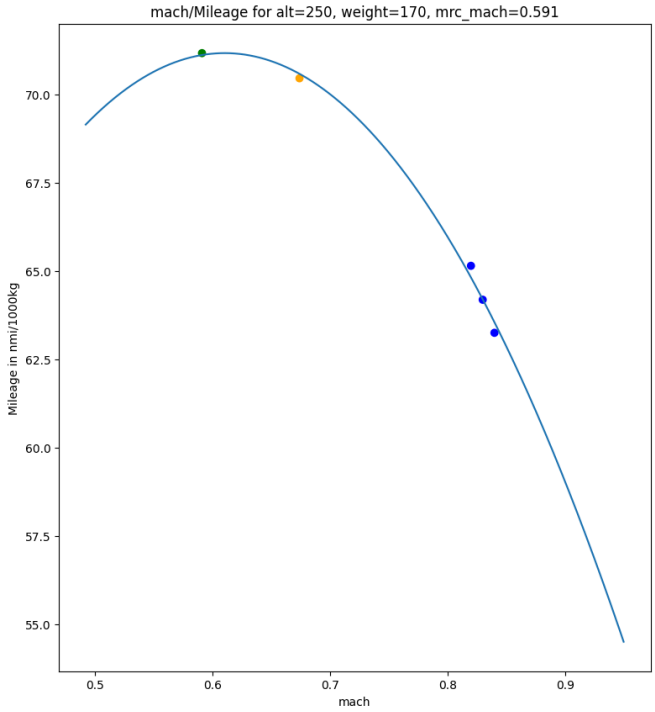
Found: 0.57 MRC and 0.653 LRC

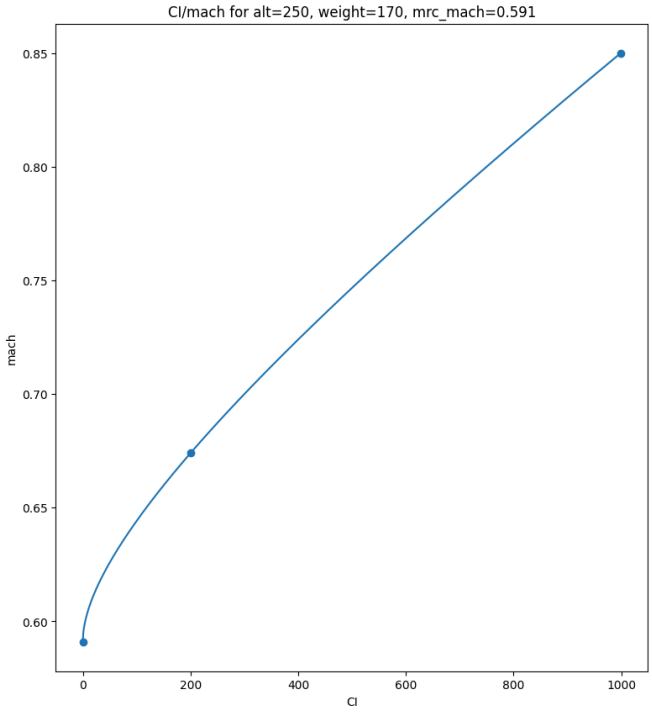




Sanity check: 0.57 MRC and 0.653 LRC

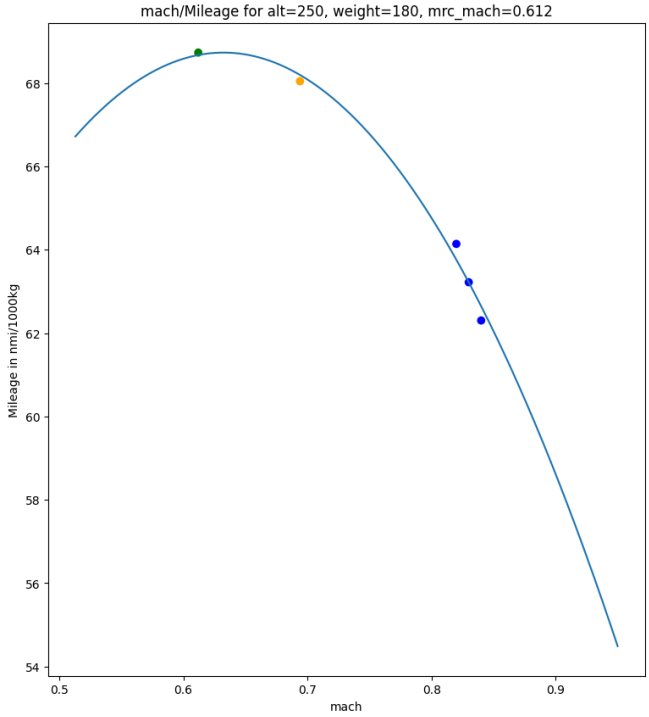
Found: 0.591 MRC and 0.674 LRC

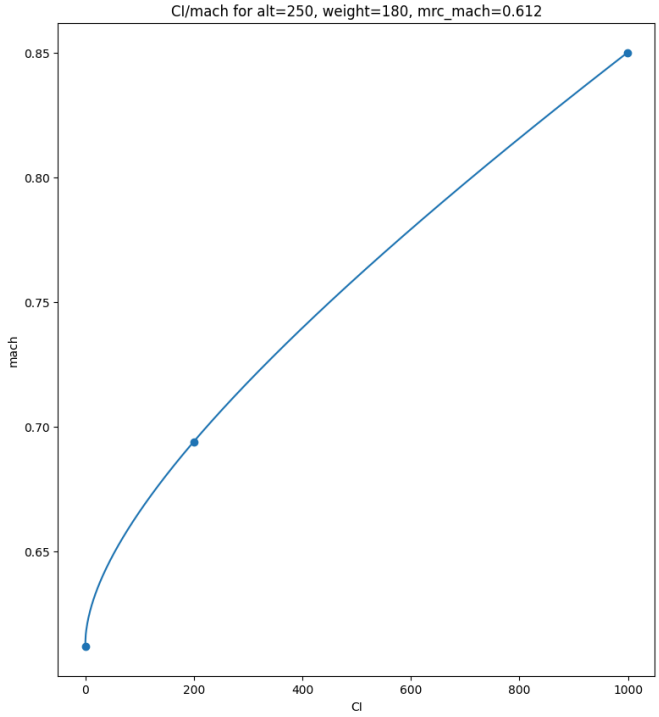




Sanity check: 0.591 MRC and 0.674 LRC

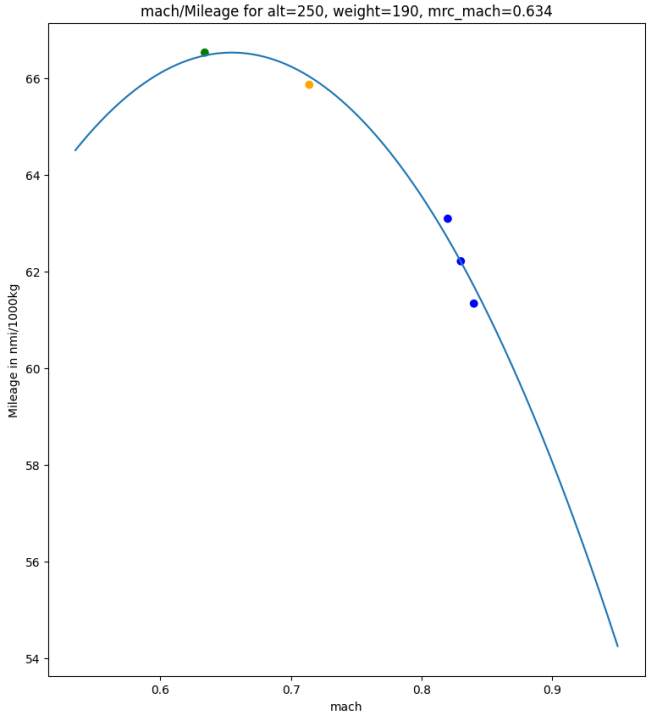
Found: 0.612 MRC and 0.694 LRC

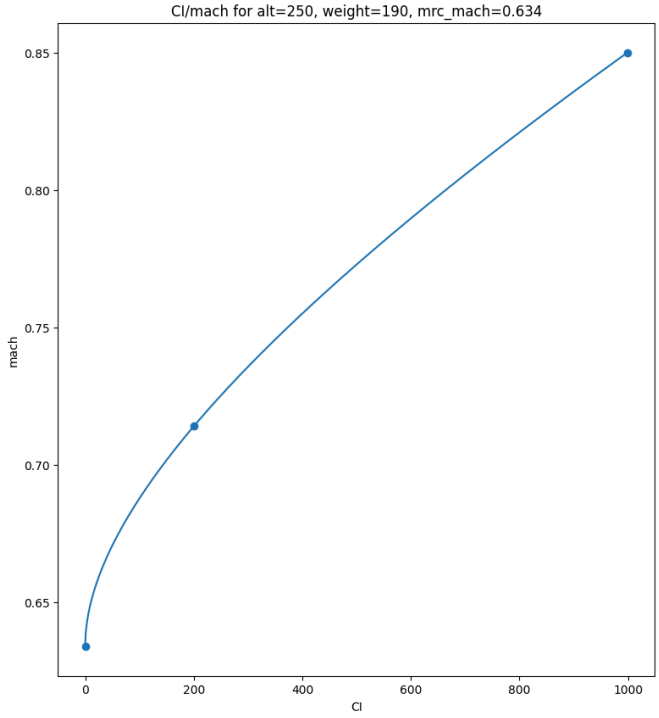




Sanity check: 0.612 MRC and 0.694 LRC

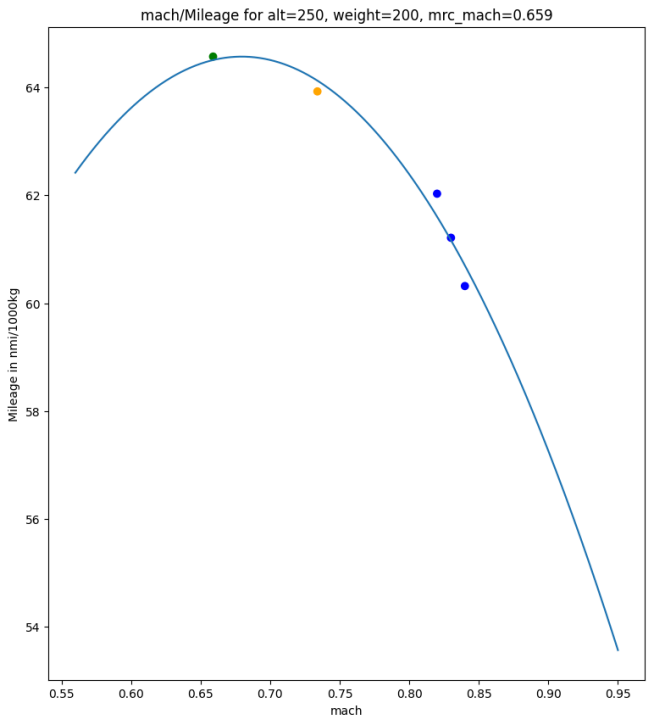
Found: 0.634 MRC and 0.714 LRC

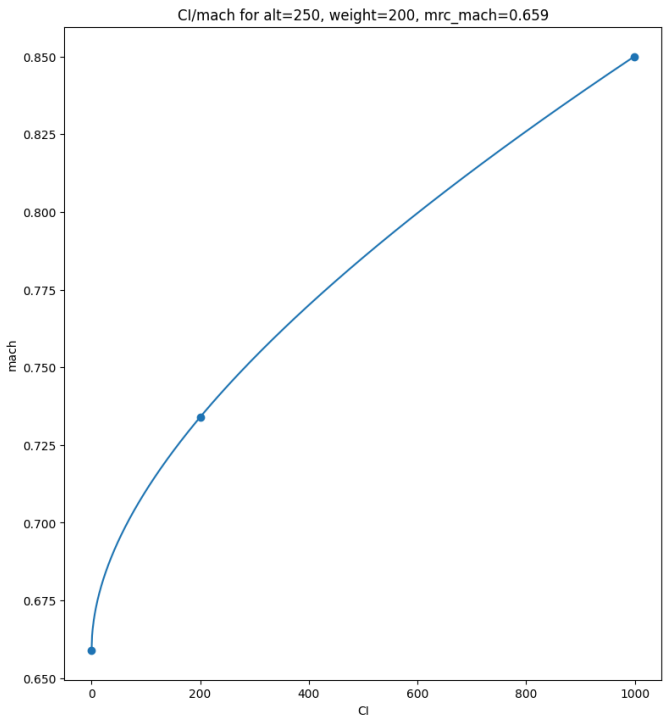




Sanity check: 0.634 MRC and 0.714 LRC

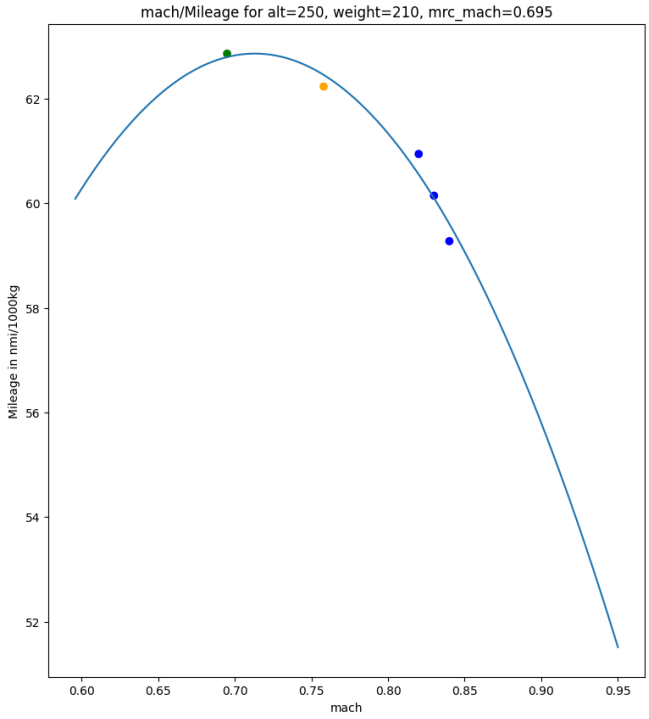
Found: 0.659 MRC and 0.734 LRC

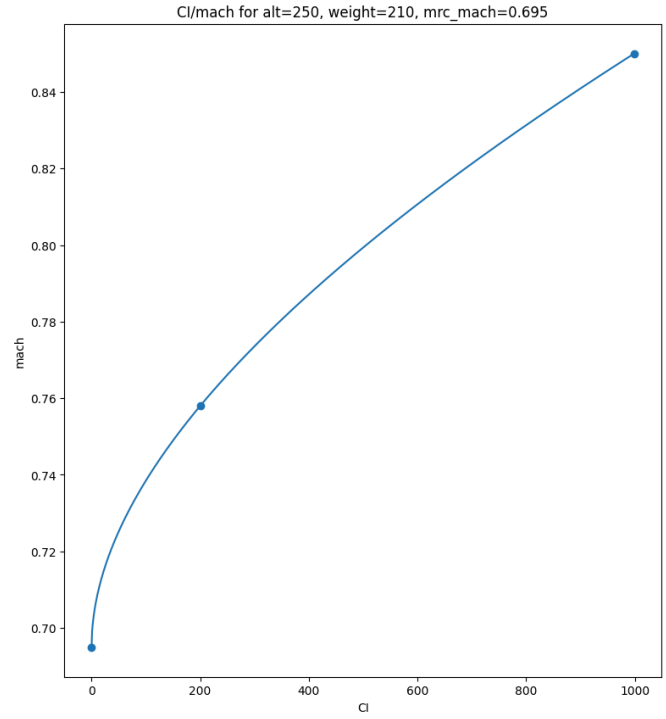




Sanity check: 0.659 MRC and 0.734 LRC

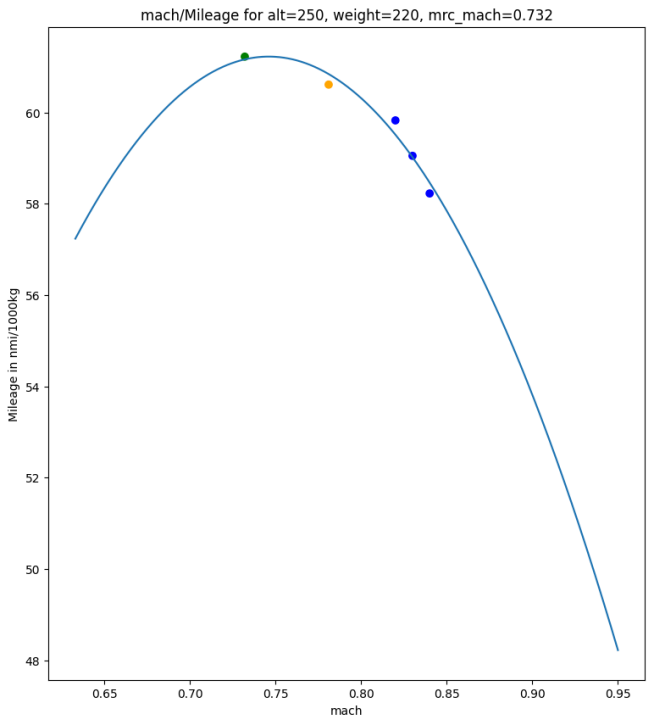
Found: 0.695 MRC and 0.758 LRC

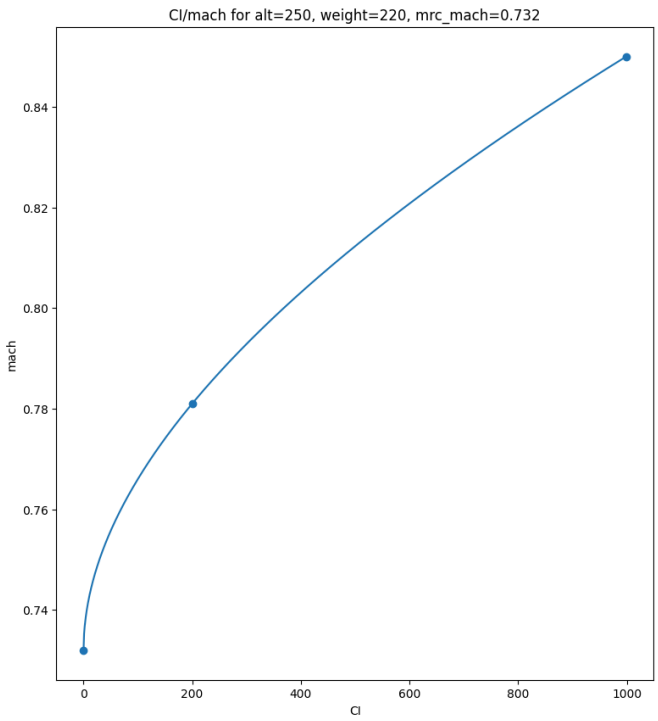




Sanity check: 0.695 MRC and 0.758 LRC

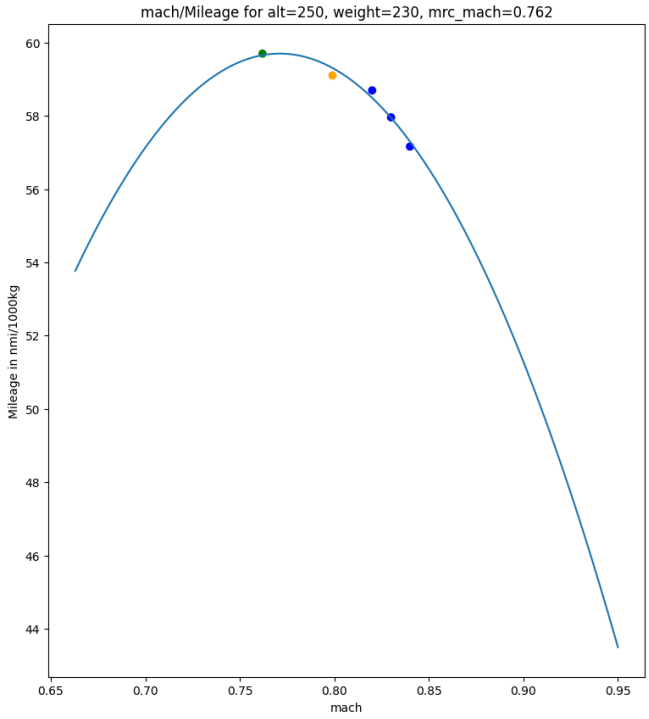
Found: 0.732 MRC and 0.781 LRC

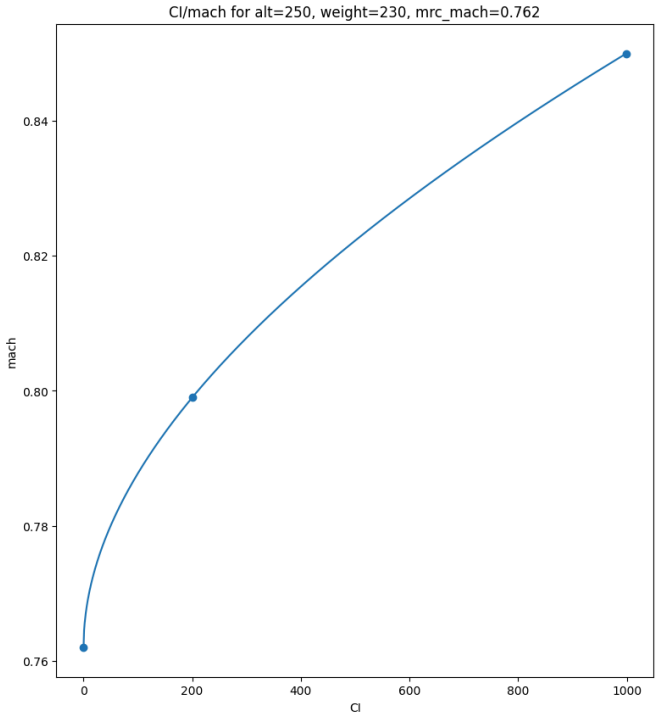




Sanity check: 0.732 MRC and 0.781 LRC

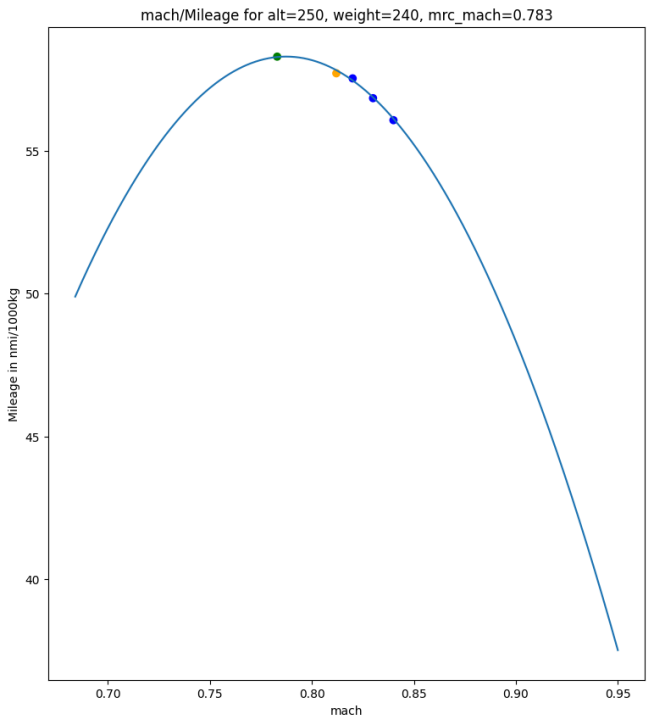
Found: 0.762 MRC and 0.799 LRC

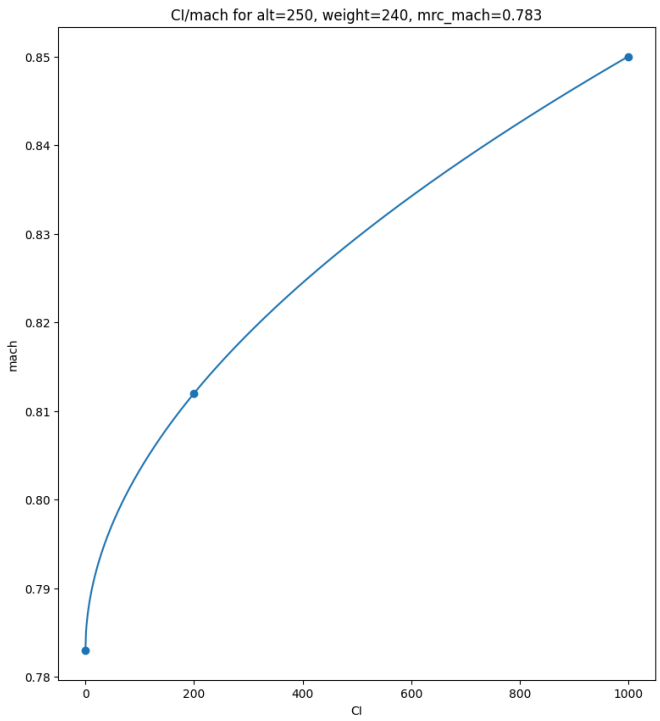




Sanity check: 0.762 MRC and 0.799 LRC

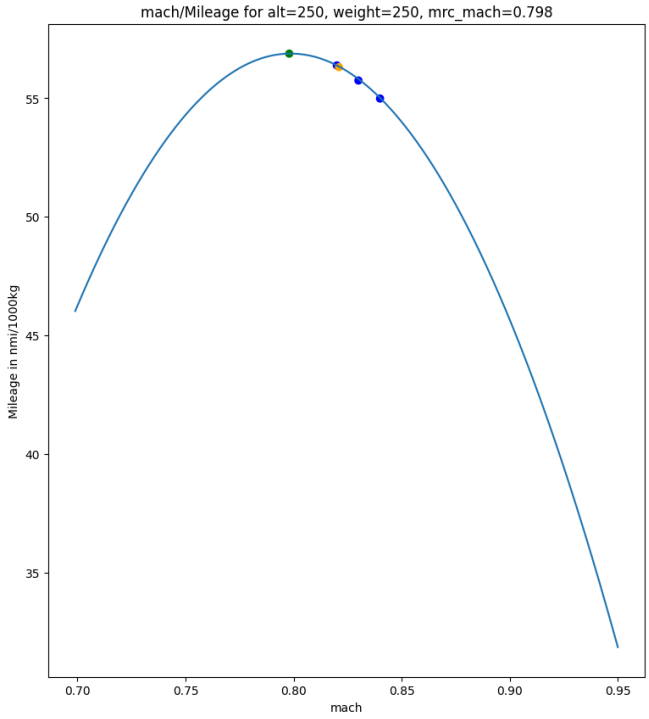
Found: 0.783 MRC and 0.812 LRC

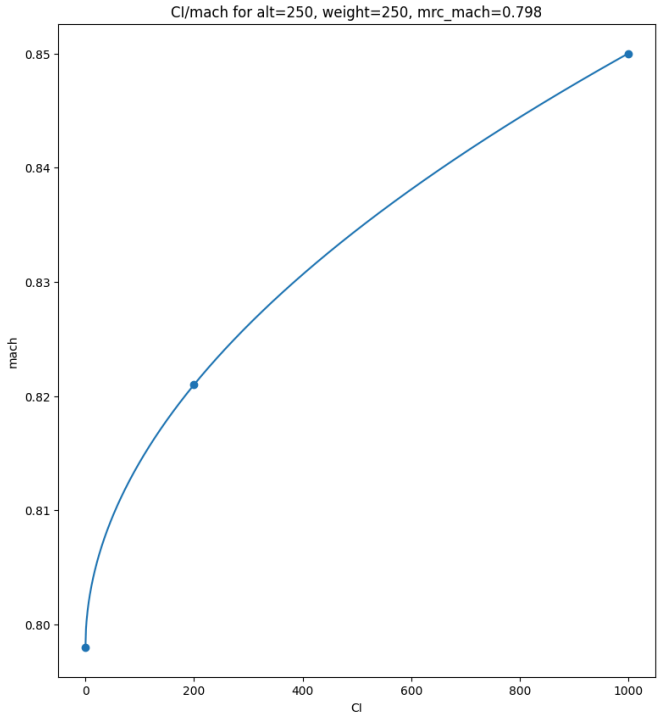




Sanity check: 0.783 MRC and 0.812 LRC

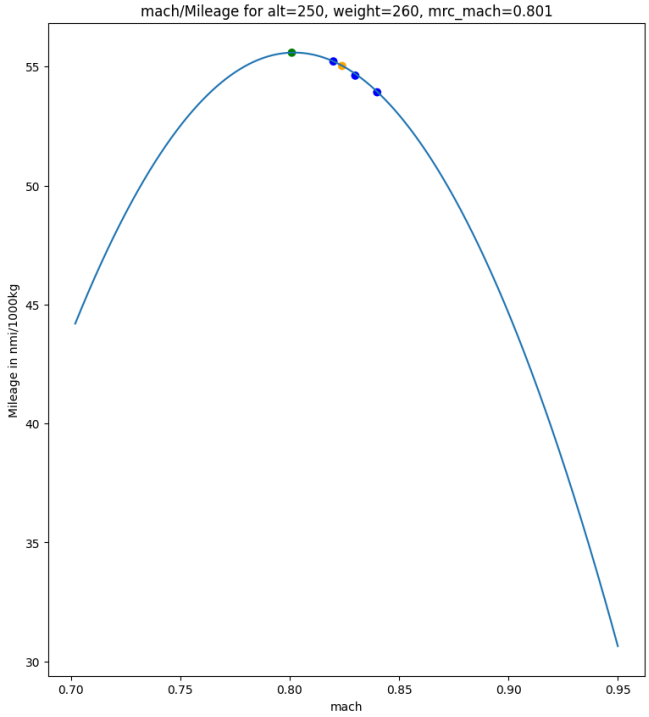
Found: 0.798 MRC and 0.821 LRC

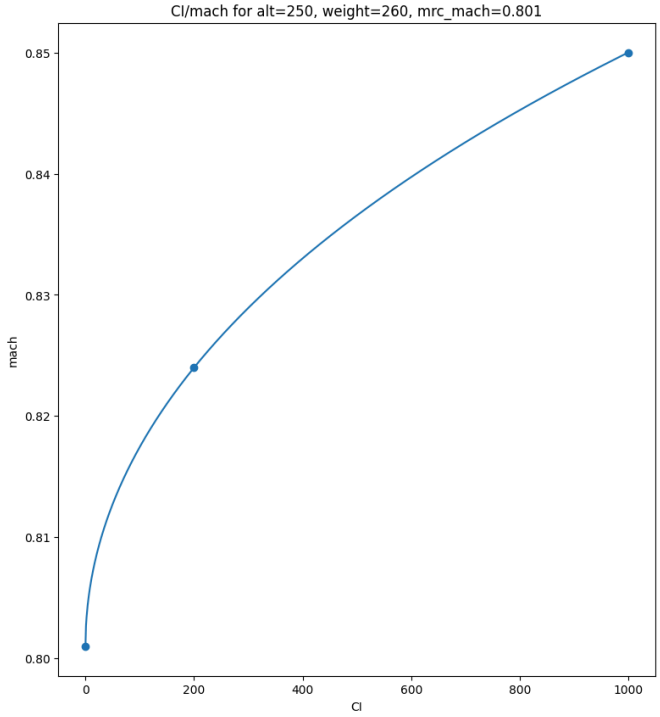




Sanity check: 0.798 MRC and 0.821 LRC

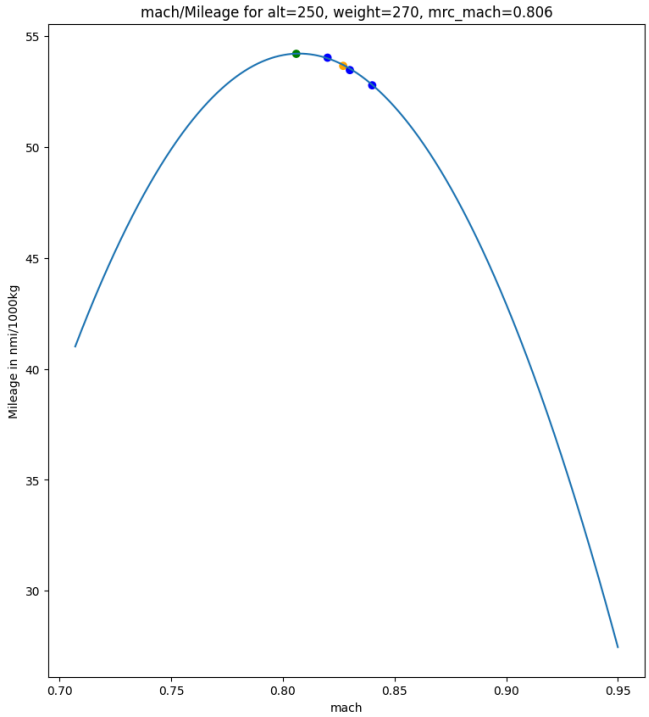
Found: 0.801 MRC and 0.824 LRC

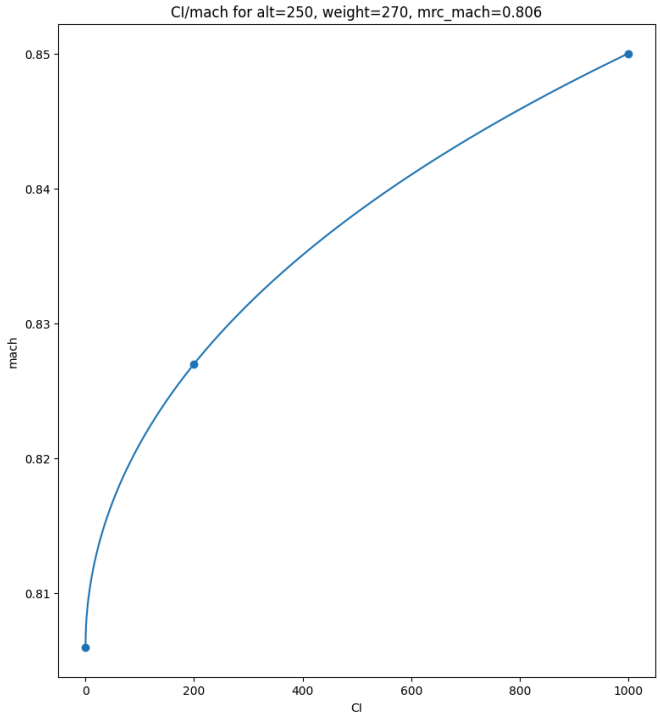




Sanity check: 0.801 MRC and 0.824 LRC

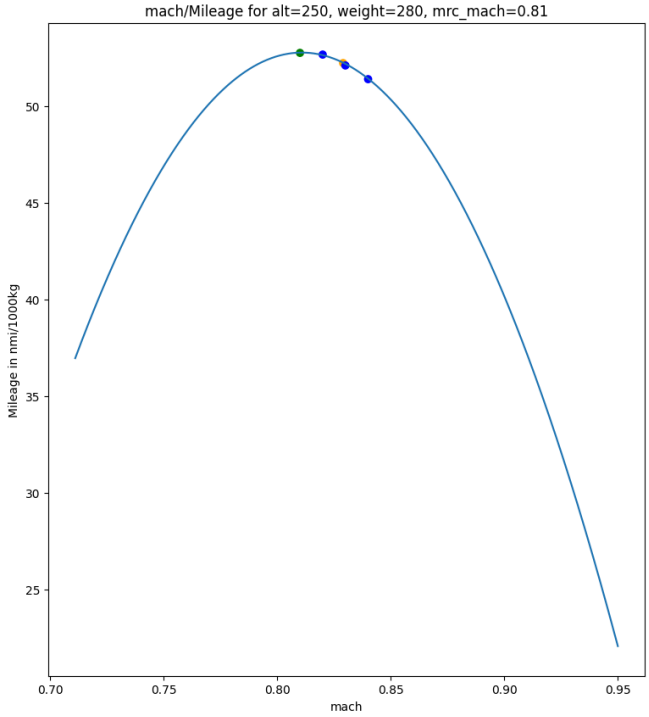
Found: 0.806 MRC and 0.827 LRC

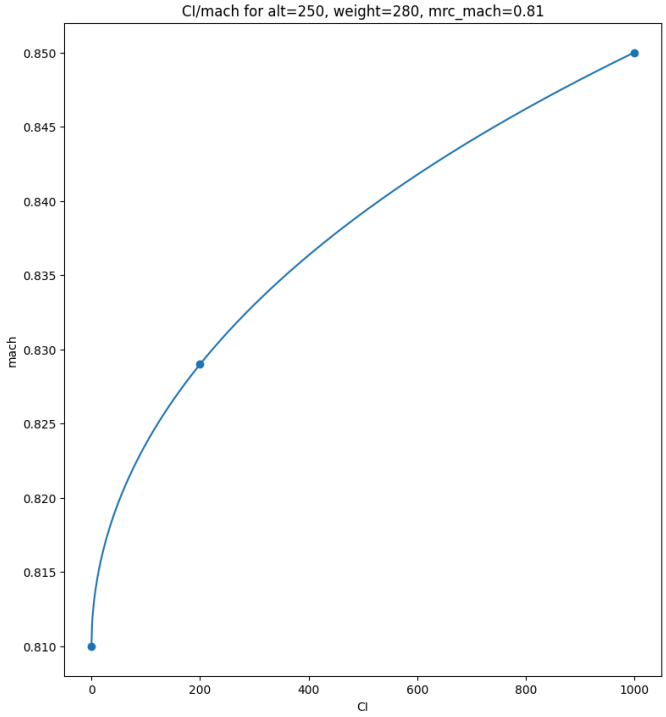




Sanity check: 0.806 MRC and 0.827 LRC

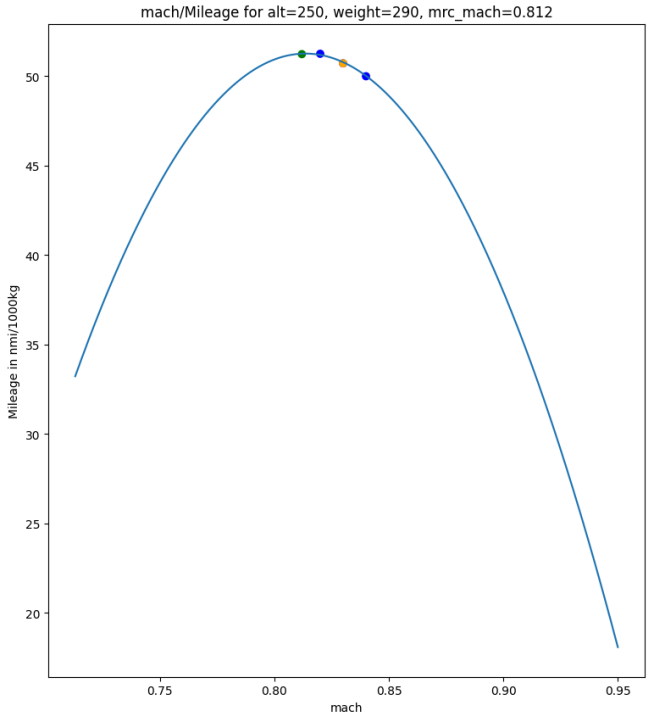
Found: 0.81 MRC and 0.829 LRC

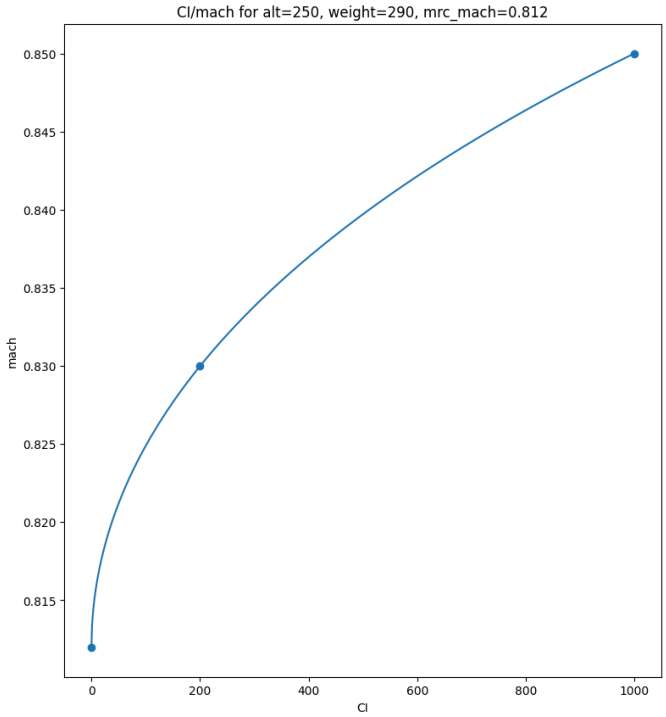




Sanity check: 0.81 MRC and 0.829 LRC

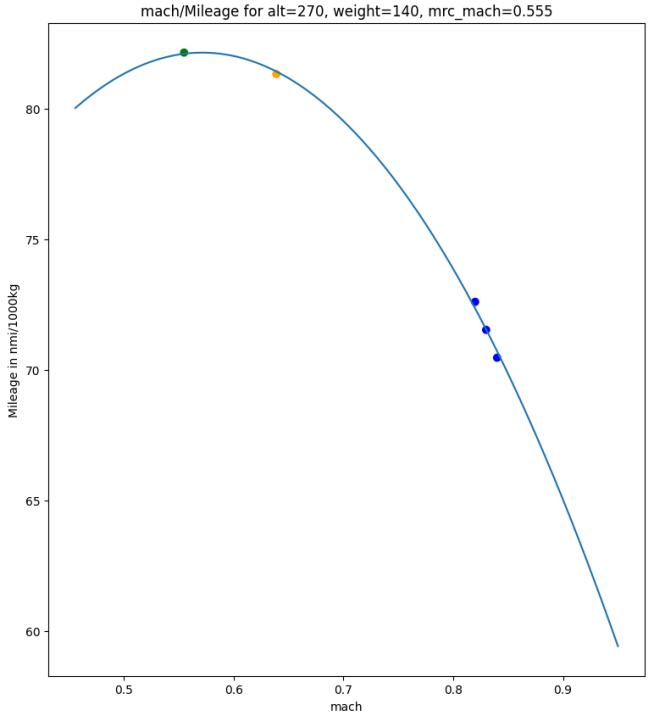
Found: 0.812 MRC and 0.83 LRC

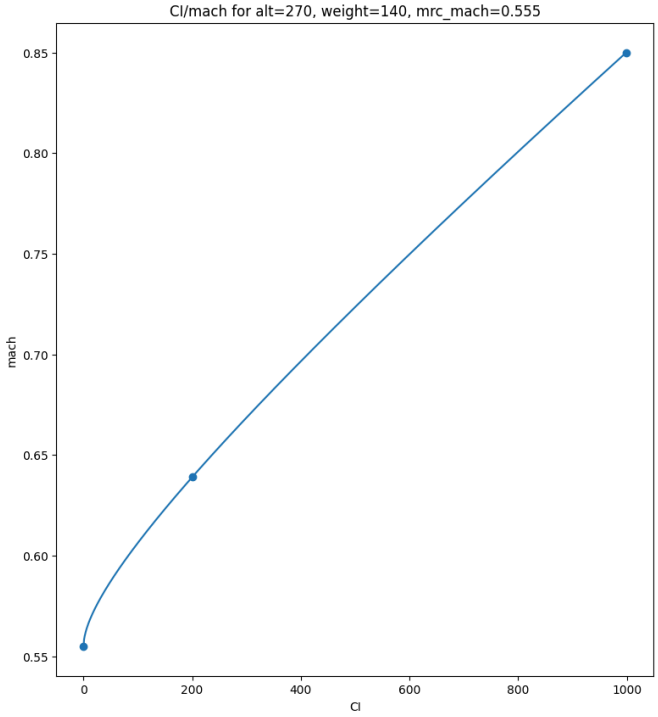




Sanity check: 0.812 MRC and 0.83 LRC

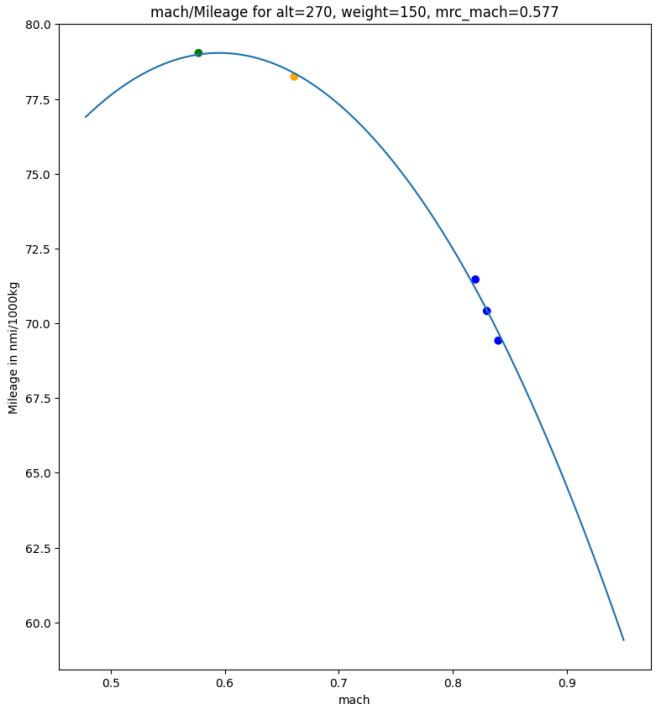
Found: 0.555 MRC and 0.639 LRC

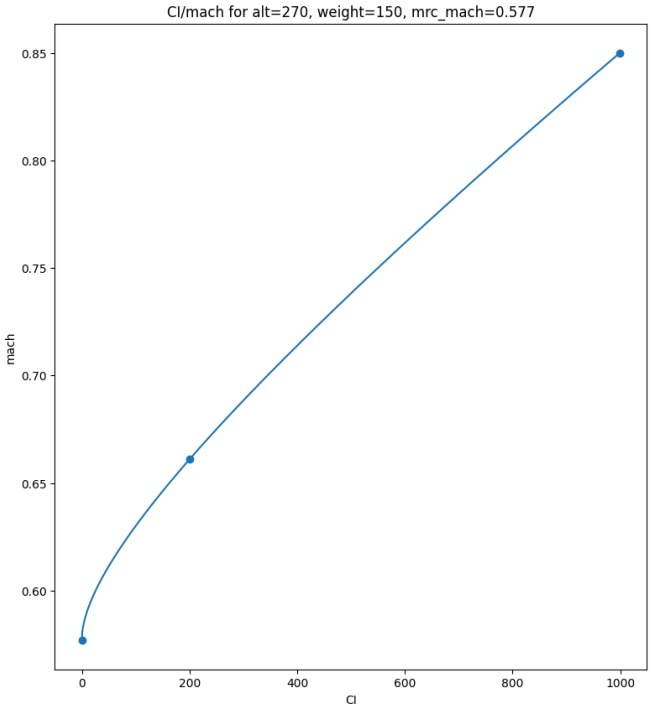




Sanity check: 0.555 MRC and 0.639 LRC

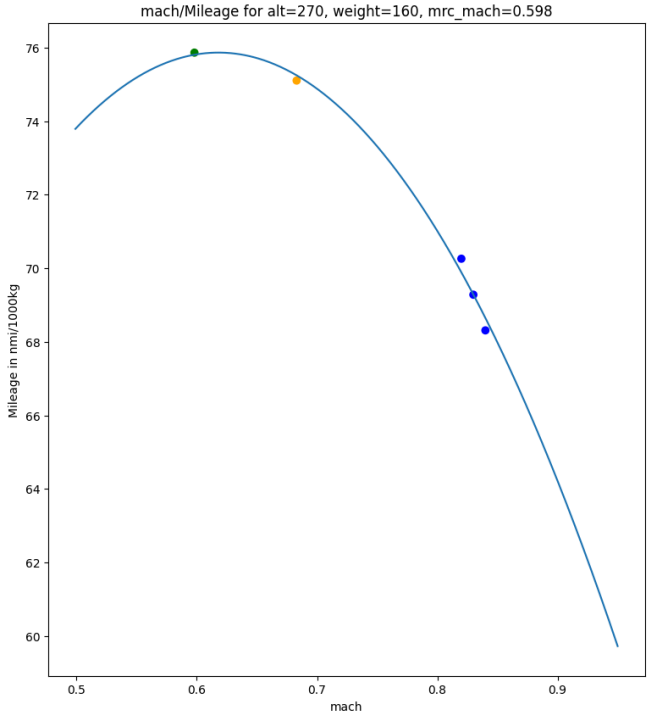
Found: 0.577 MRC and 0.661 LRC

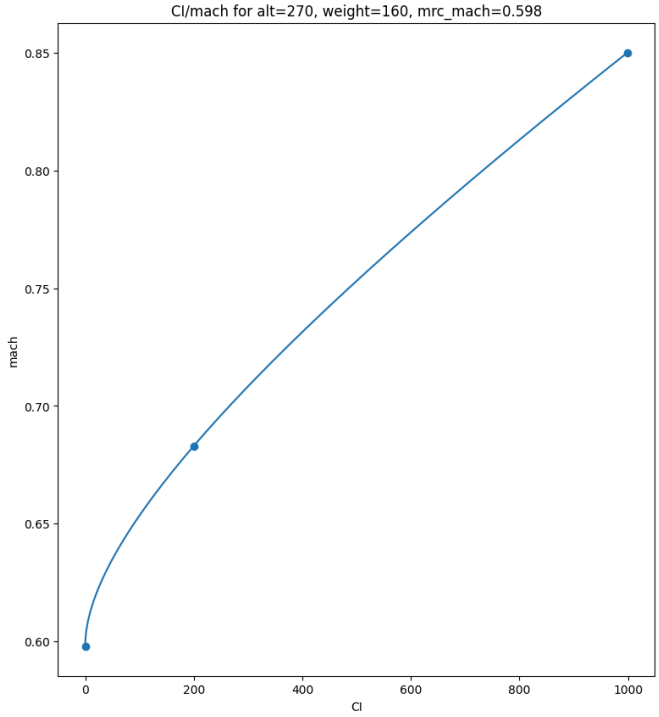




Sanity check: 0.577 MRC and 0.661 LRC

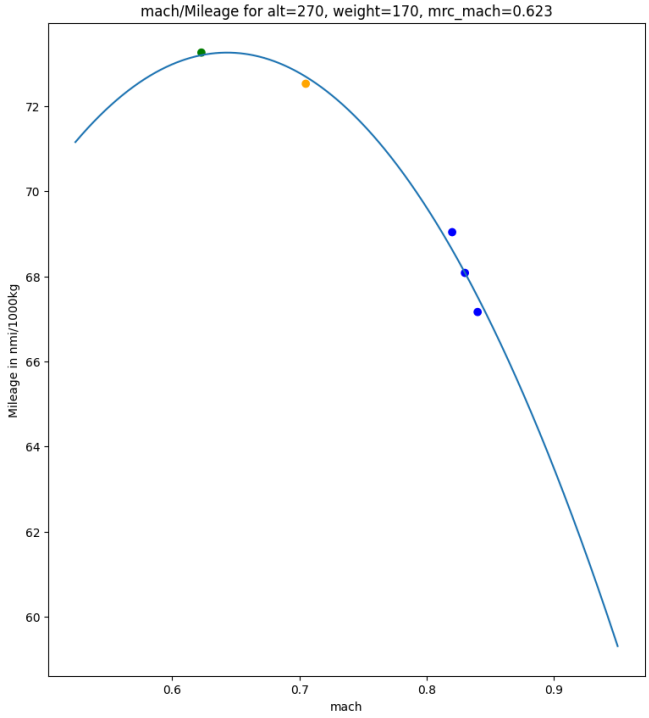
Found: 0.598 MRC and 0.683 LRC

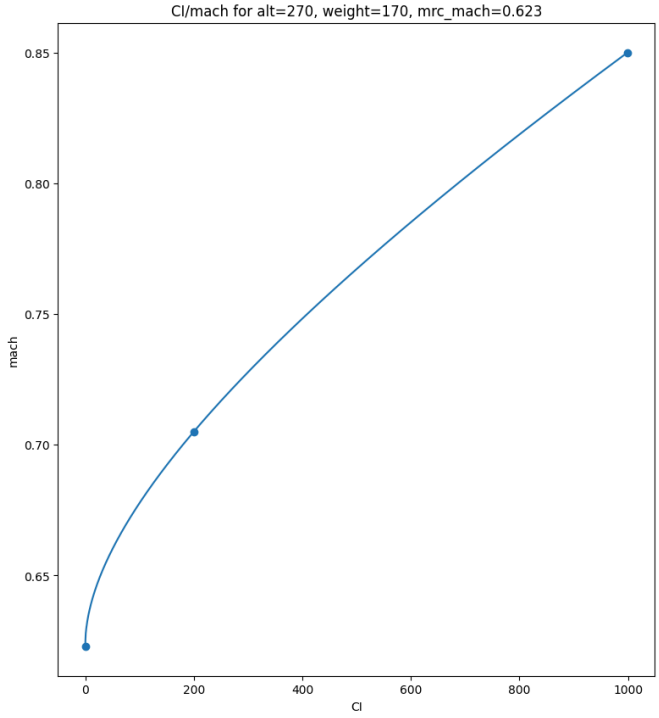




Sanity check: 0.598 MRC and 0.683 LRC

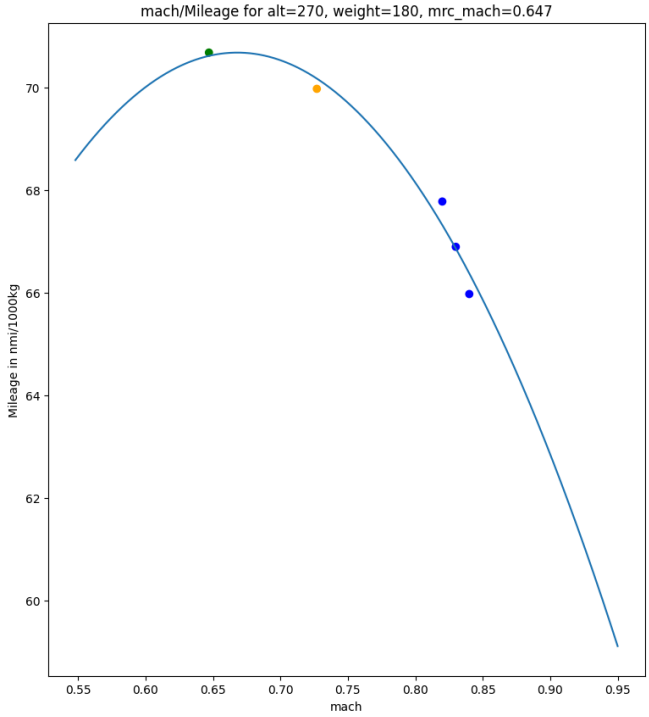
Found: 0.623 MRC and 0.705 LRC

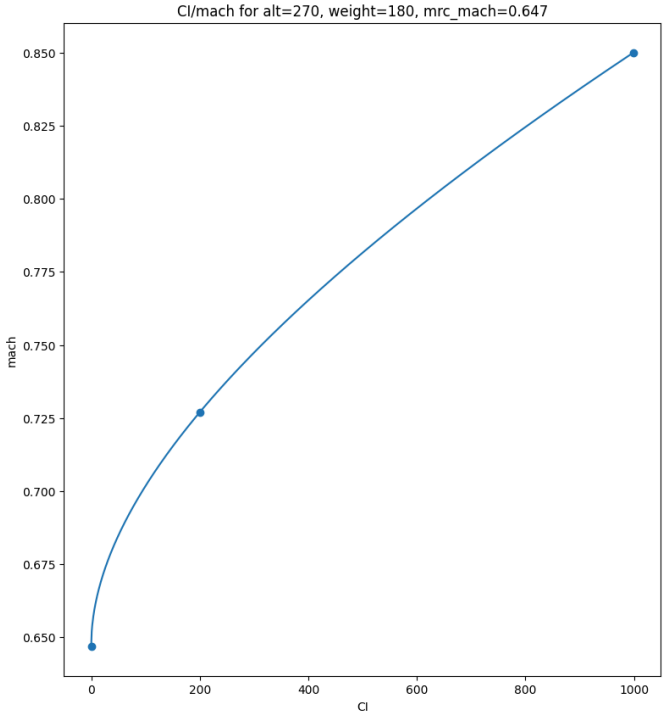




Sanity check: 0.623 MRC and 0.705 LRC

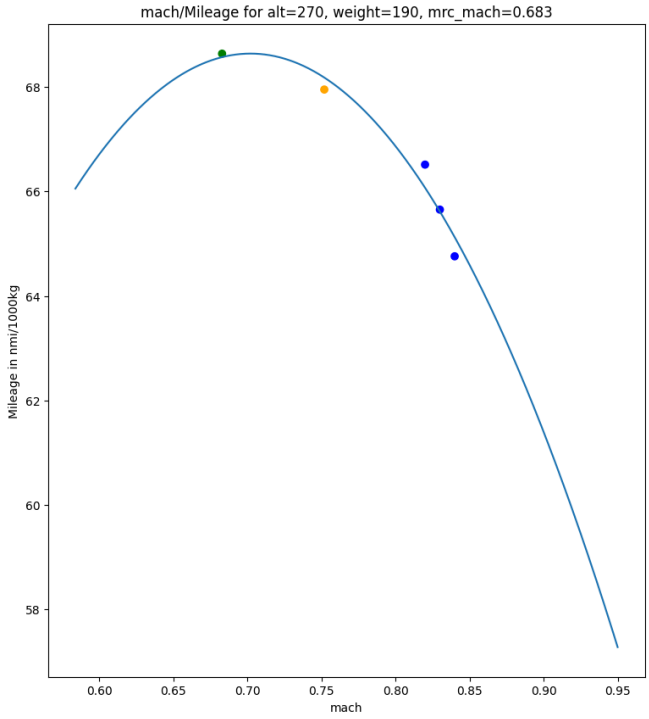
Found: 0.647 MRC and 0.727 LRC

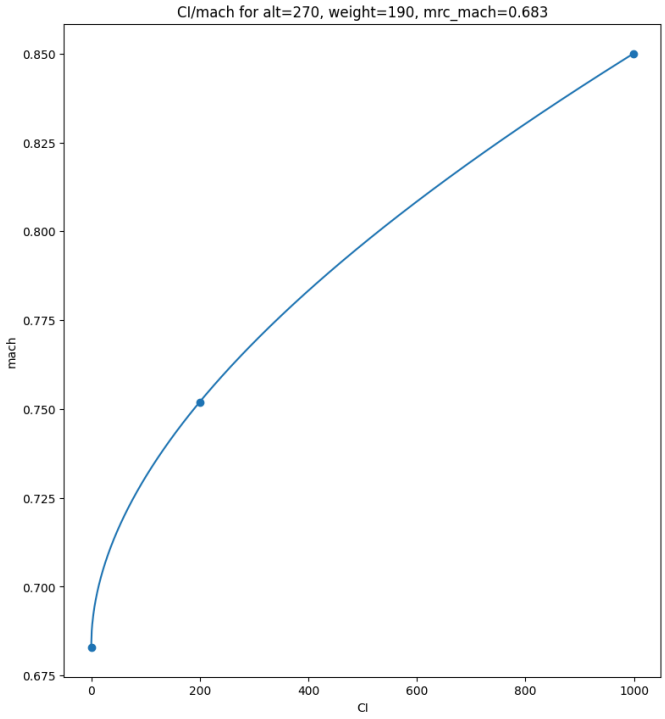




Sanity check: 0.647 MRC and 0.727 LRC

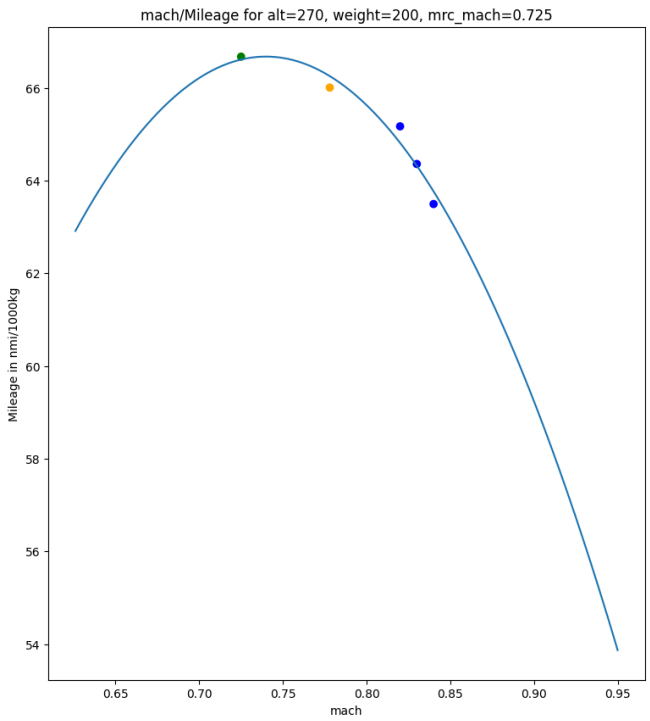
Found: 0.683 MRC and 0.752 LRC

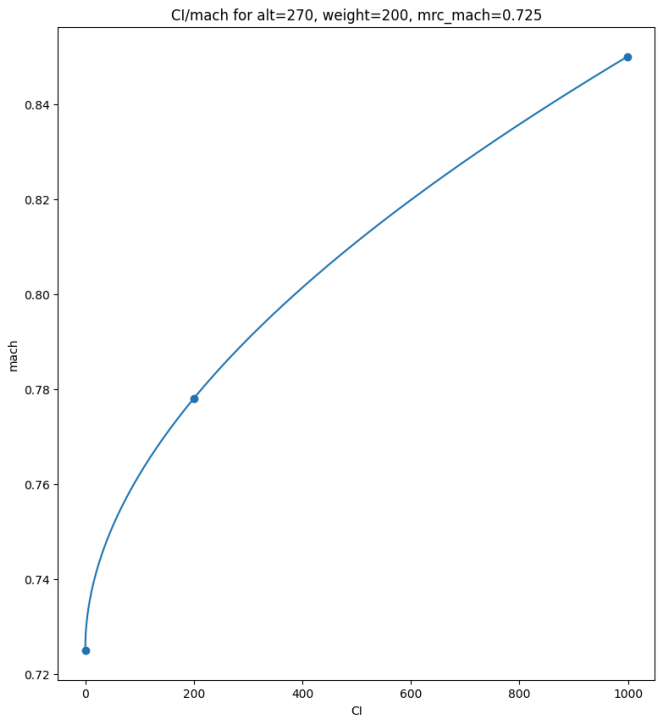




Sanity check: 0.683 MRC and 0.752 LRC

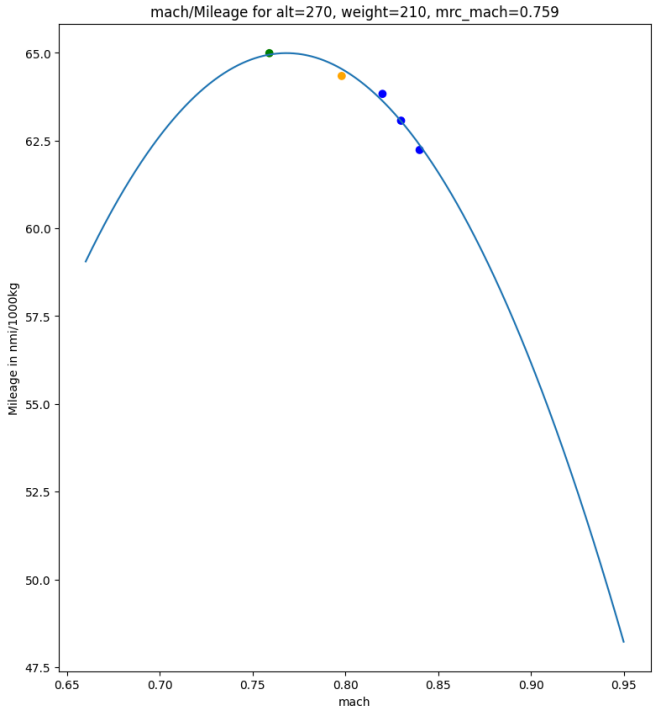
Found: 0.725 MRC and 0.778 LRC

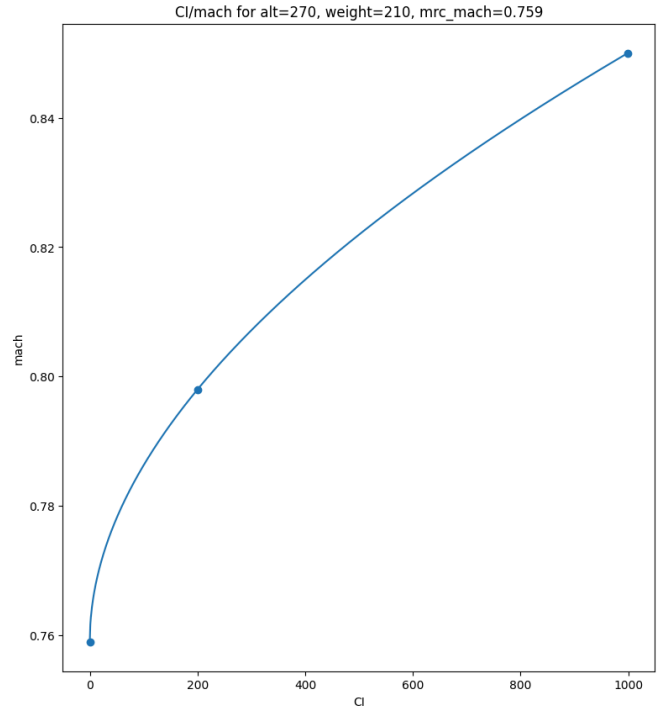




Sanity check: 0.725 MRC and 0.778 LRC

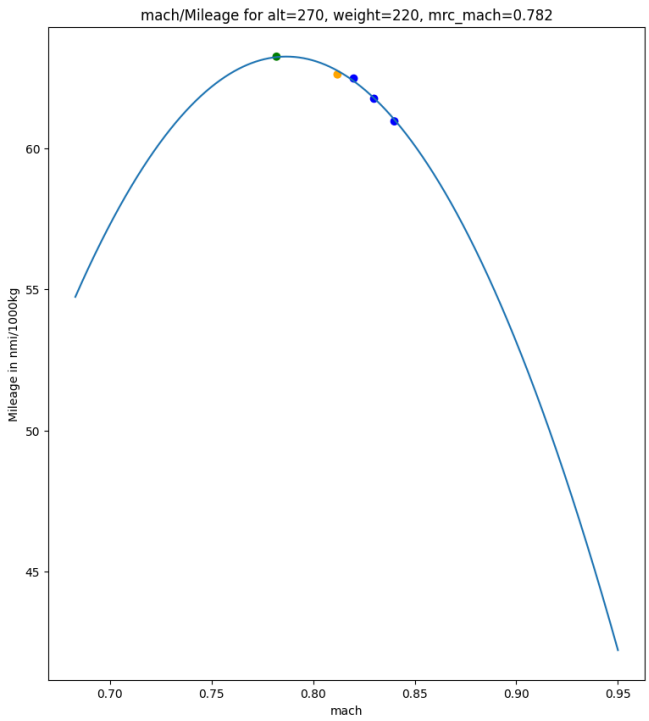
Found: 0.759 MRC and 0.798 LRC

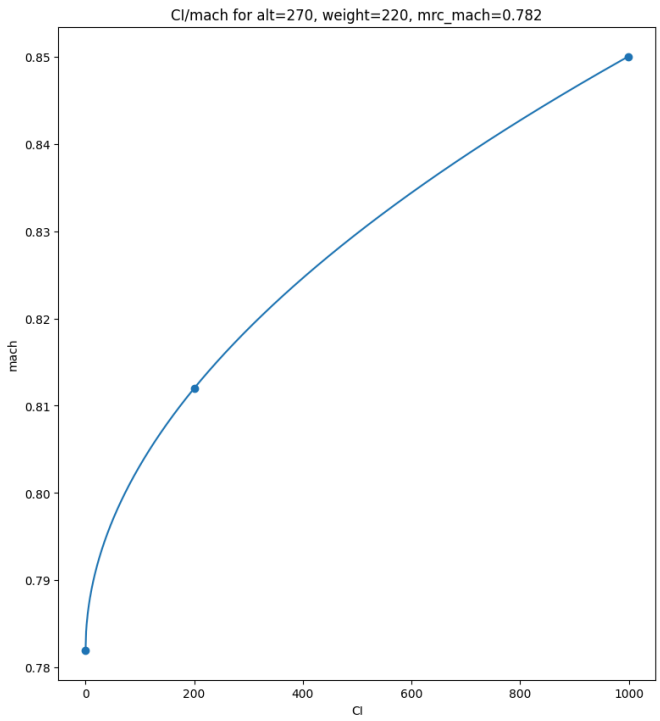




Sanity check: 0.759 MRC and 0.798 LRC

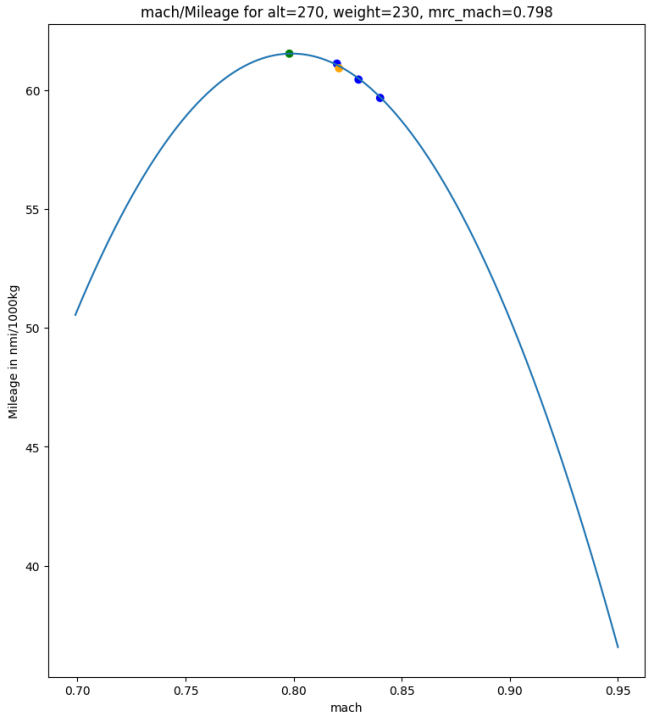
Found: 0.782 MRC and 0.812 LRC

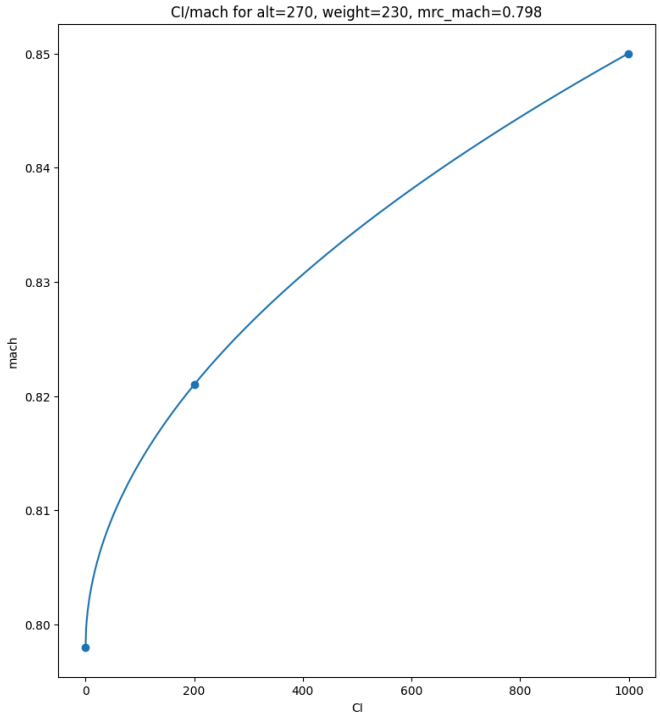




Sanity check: 0.782 MRC and 0.812 LRC

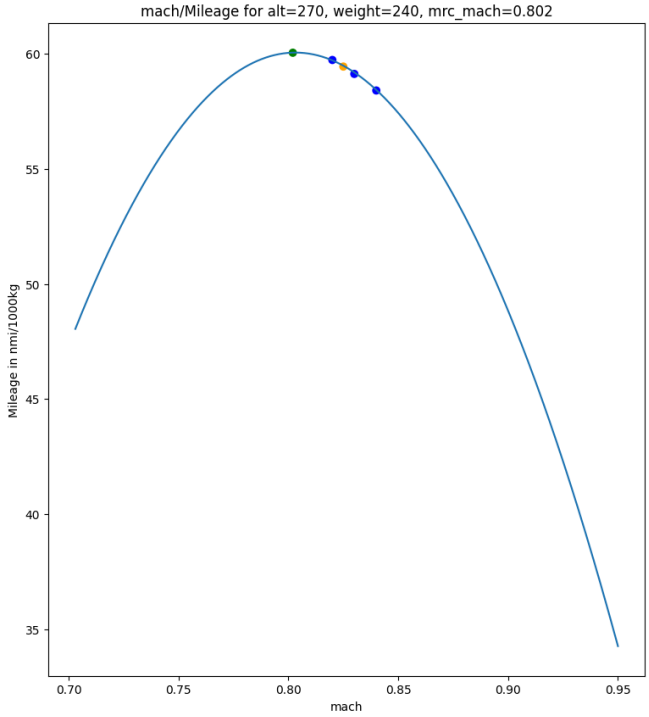
Found: 0.798 MRC and 0.821 LRC

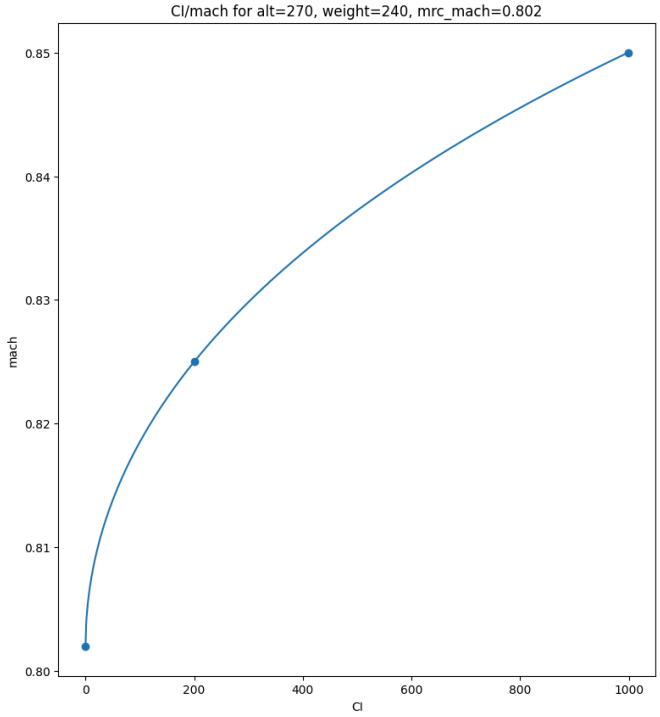




Sanity check: 0.798 MRC and 0.821 LRC

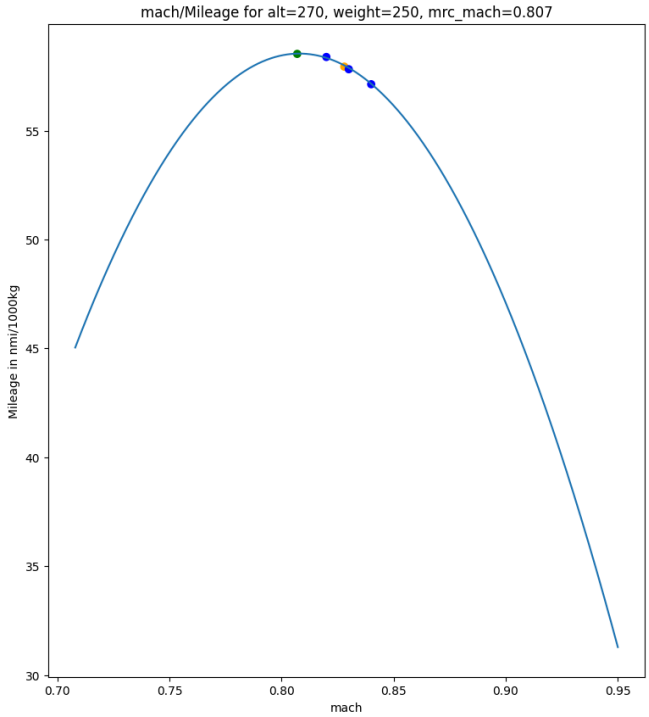
Found: 0.802 MRC and 0.825 LRC

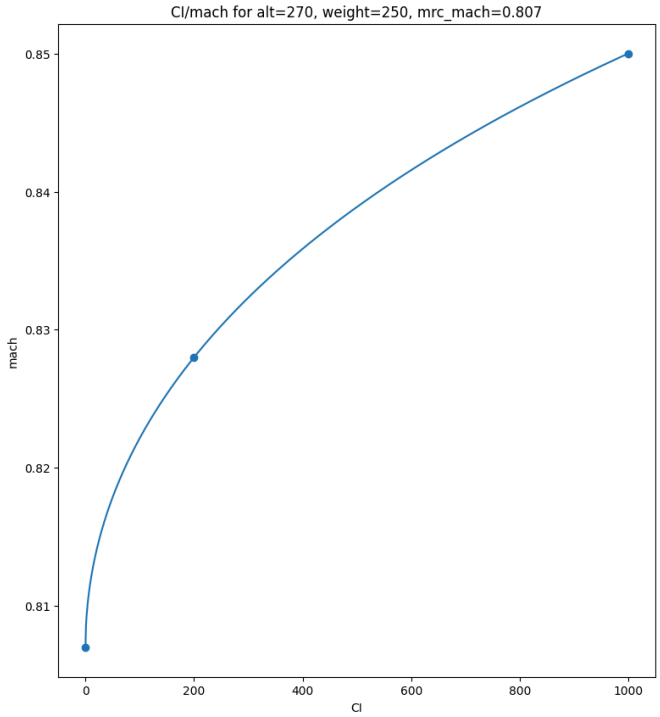




Sanity check: 0.802 MRC and 0.825 LRC

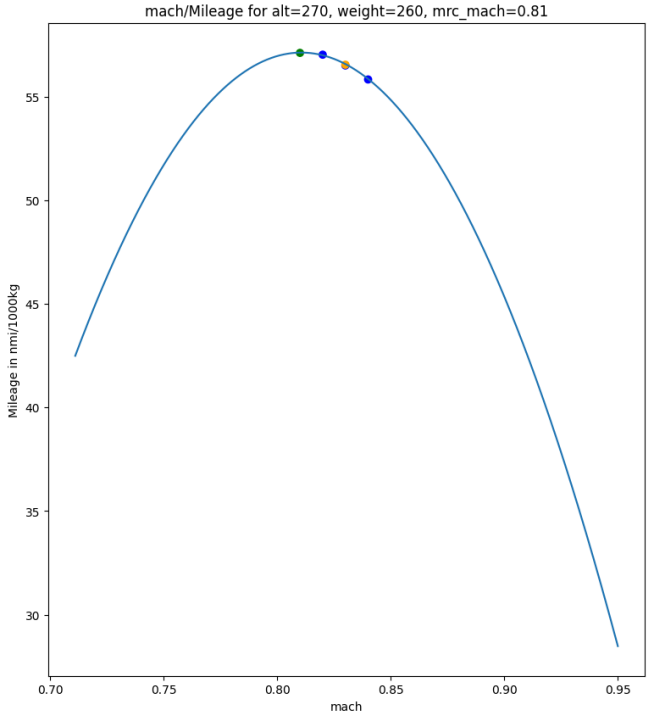
Found: 0.807 MRC and 0.828 LRC

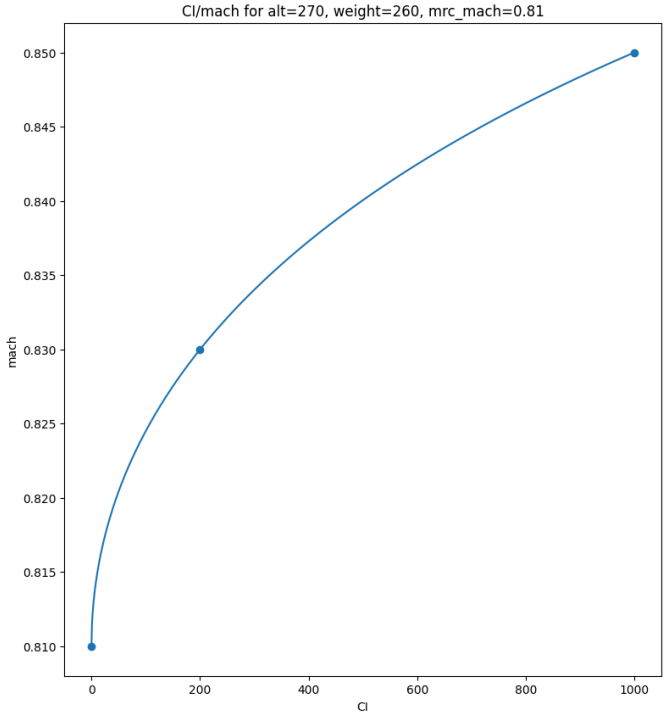




Sanity check: 0.807 MRC and 0.828 LRC

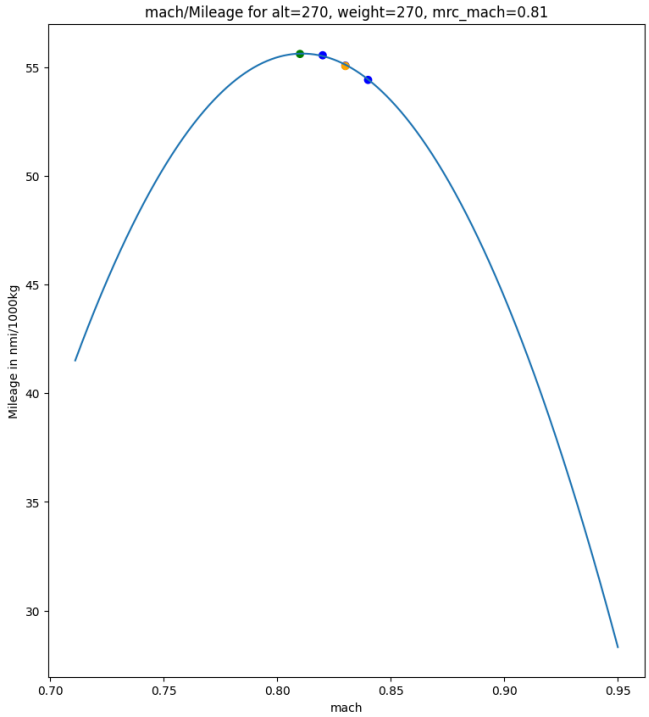
Found: 0.81 MRC and 0.83 LRC

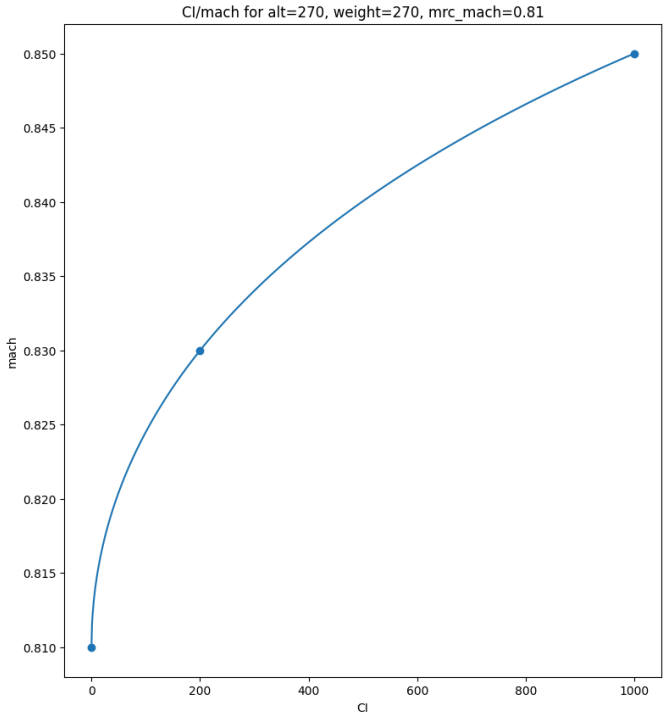




Sanity check: 0.81 MRC and 0.83 LRC

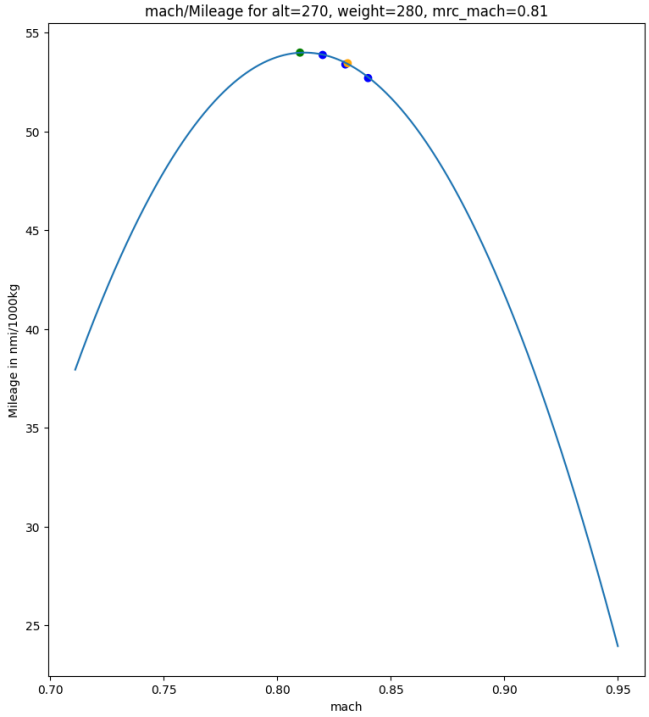
Found: 0.81 MRC and 0.83 LRC

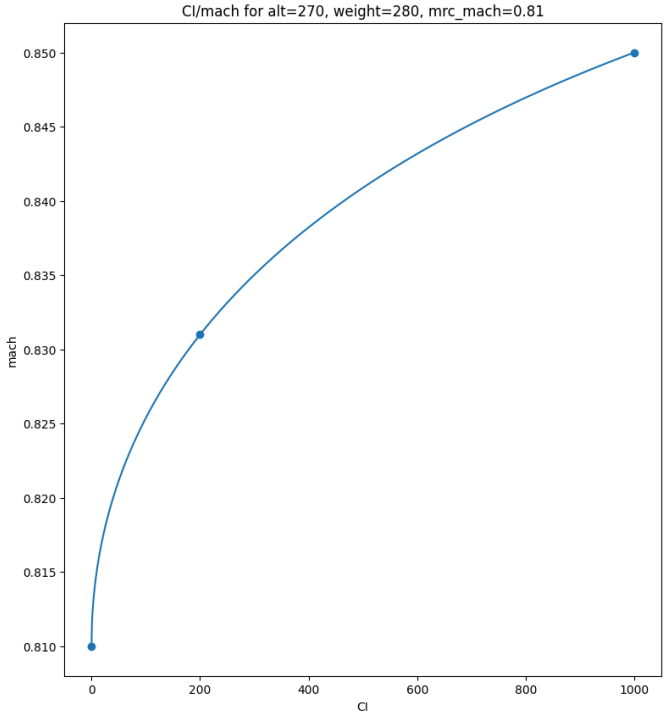




Sanity check: 0.81 MRC and 0.83 LRC

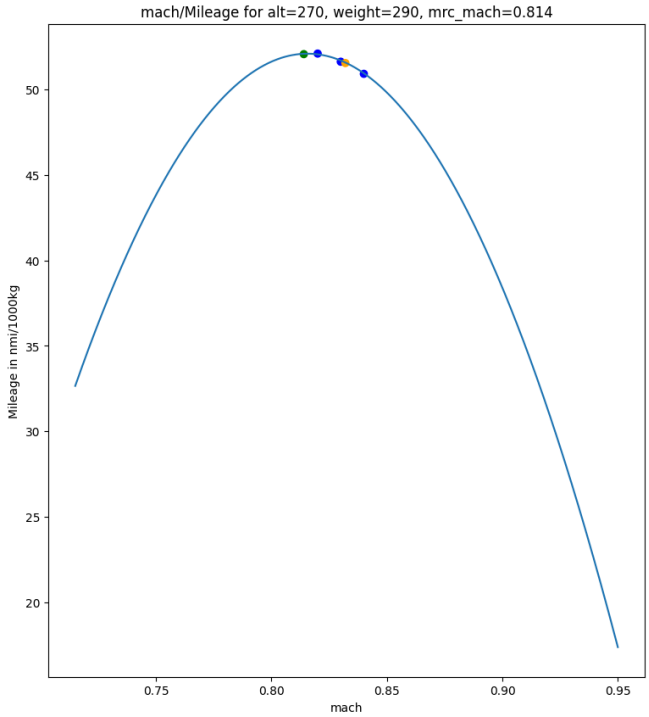
Found: 0.81 MRC and 0.831 LRC

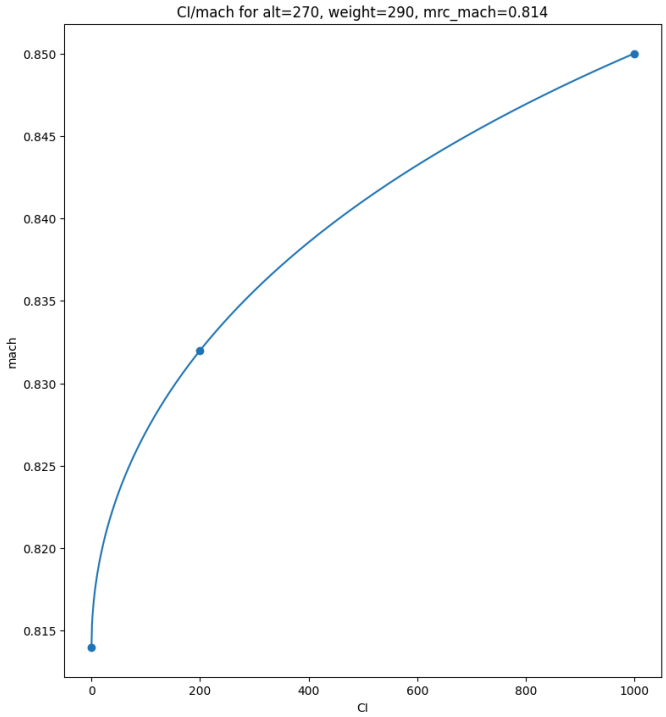




Sanity check: 0.81 MRC and 0.831 LRC

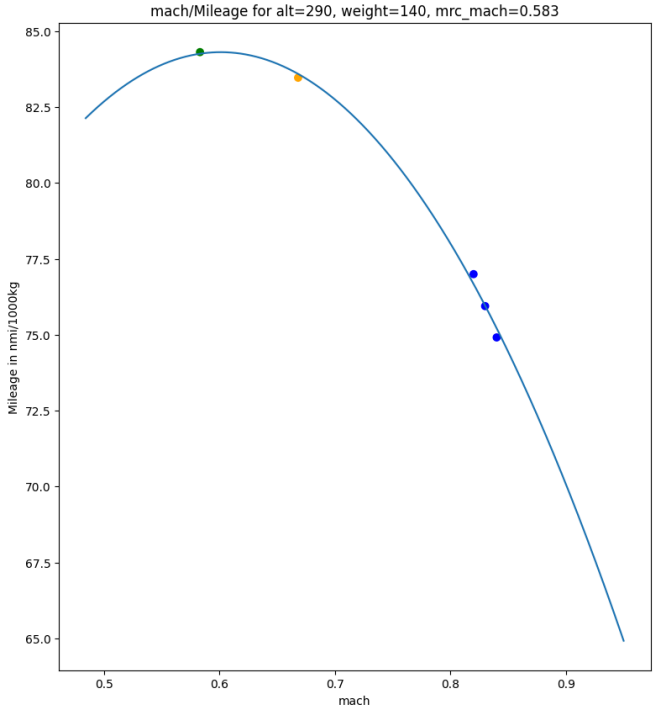
Found: 0.814 MRC and 0.832 LRC

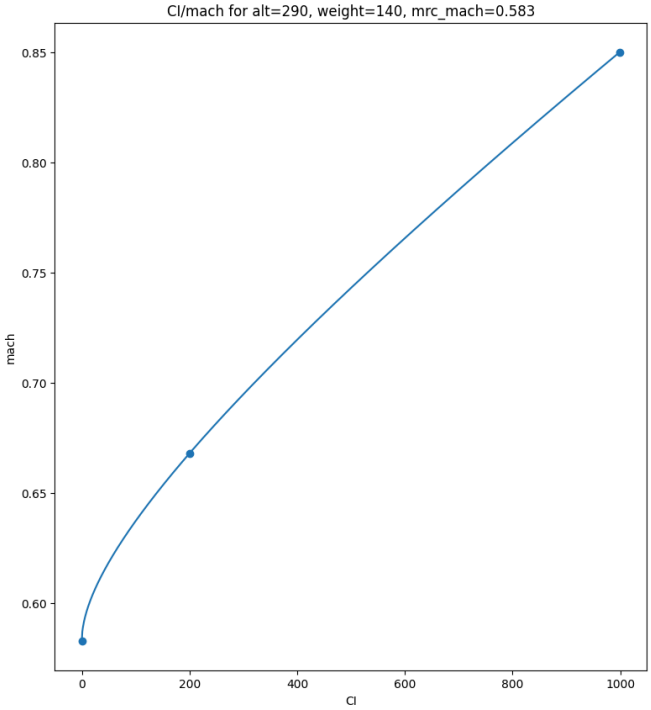




Sanity check: 0.814 MRC and 0.832 LRC

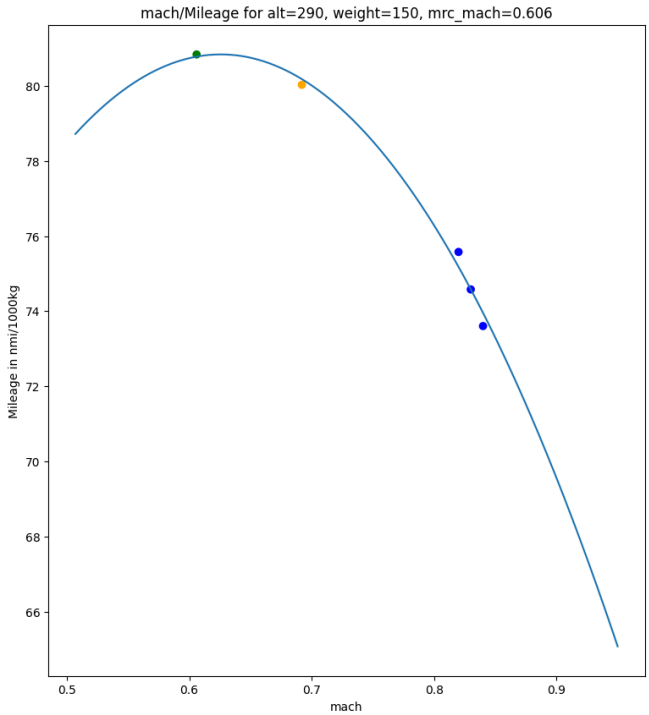
Found: 0.583 MRC and 0.668 LRC

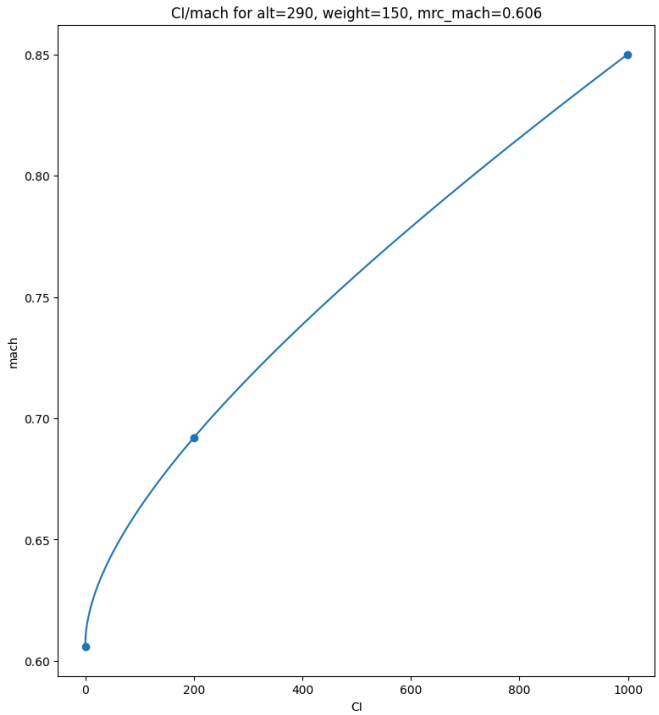




Sanity check: 0.583 MRC and 0.668 LRC

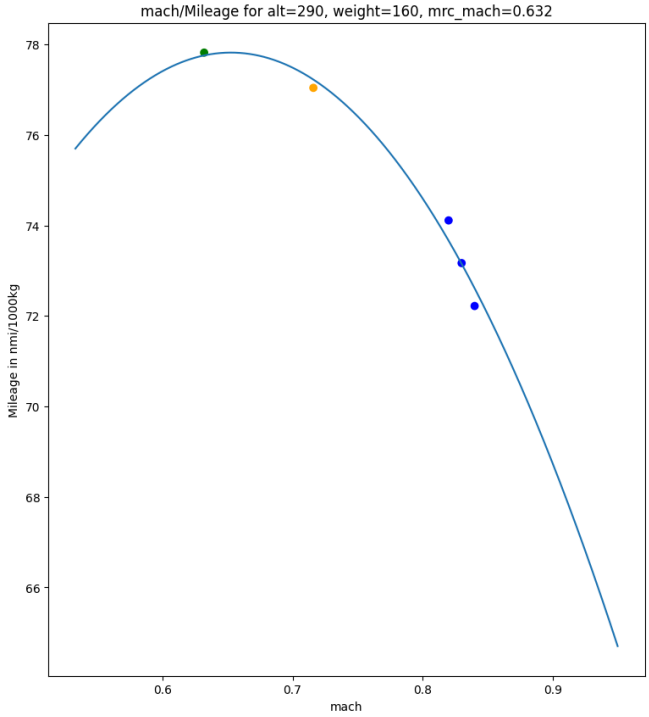
Found: 0.606 MRC and 0.692 LRC

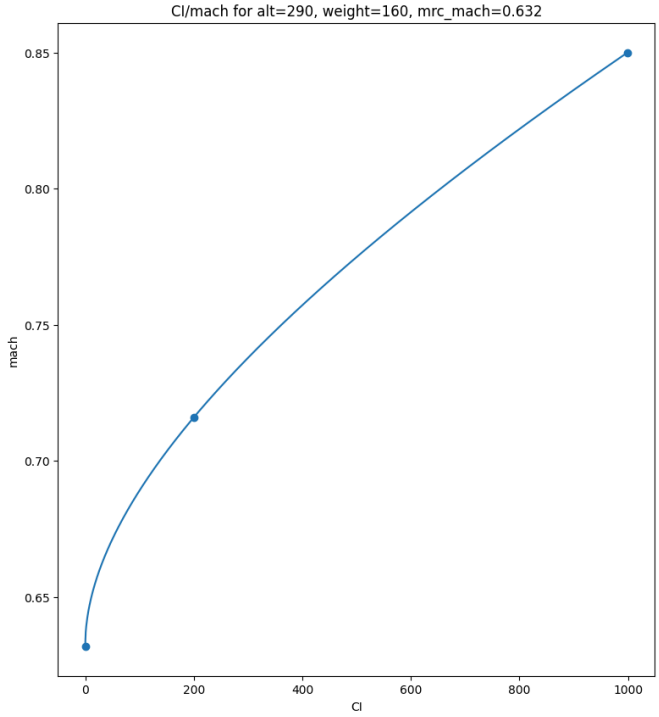




Sanity check: 0.606 MRC and 0.692 LRC

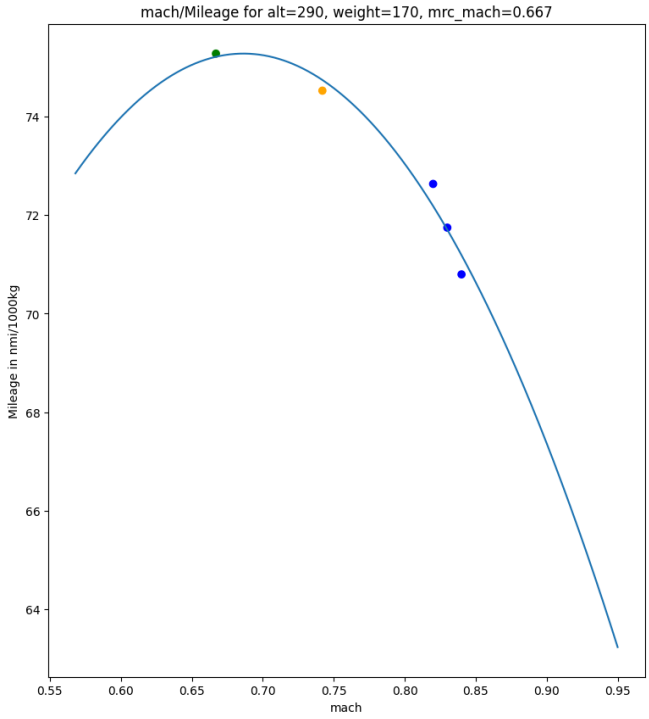
Found: 0.632 MRC and 0.716 LRC

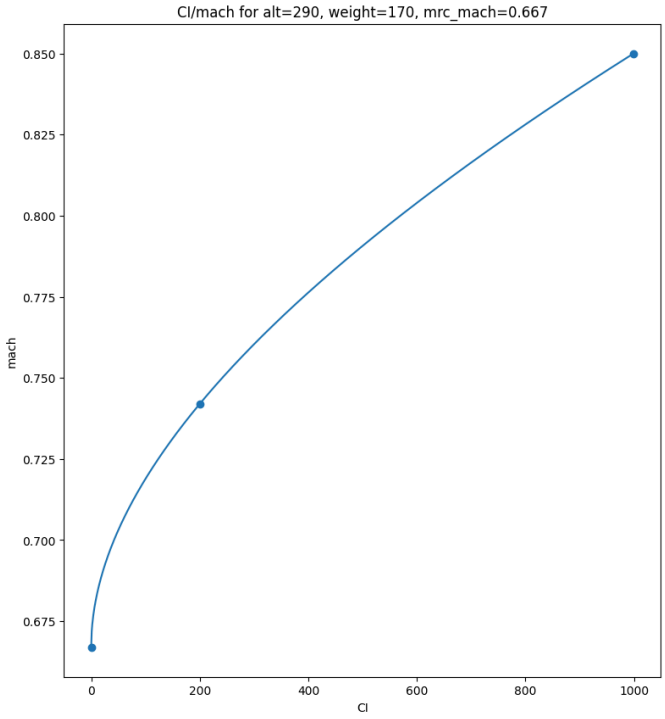




Sanity check: 0.632 MRC and 0.716 LRC

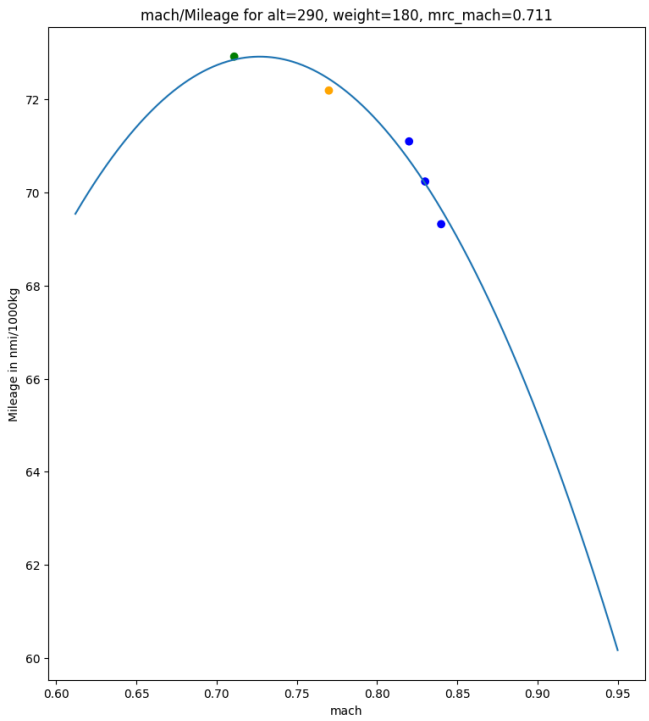
Found: 0.667 MRC and 0.742 LRC

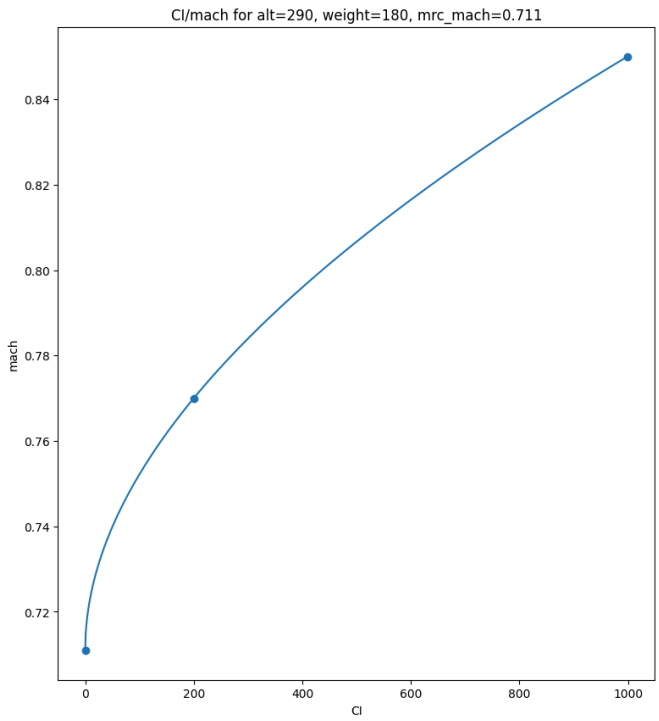




Sanity check: 0.667 MRC and 0.742 LRC

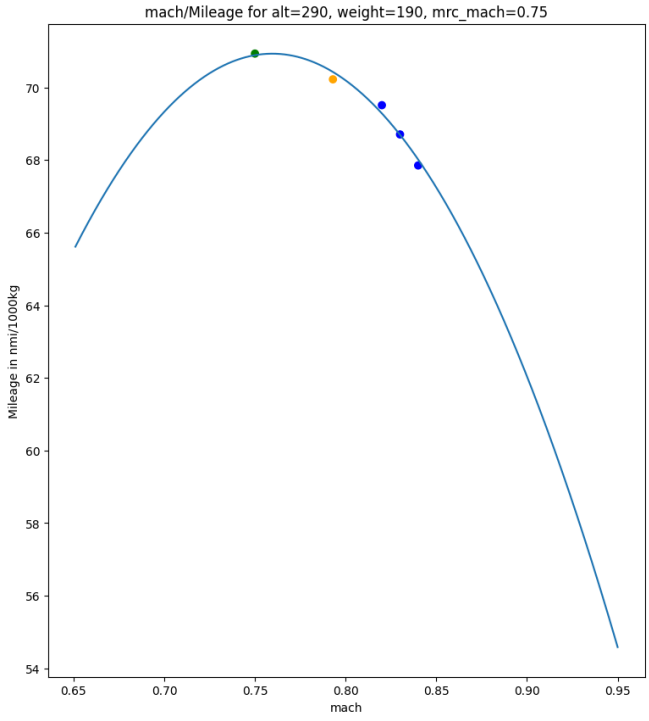
Found: 0.711 MRC and 0.77 LRC

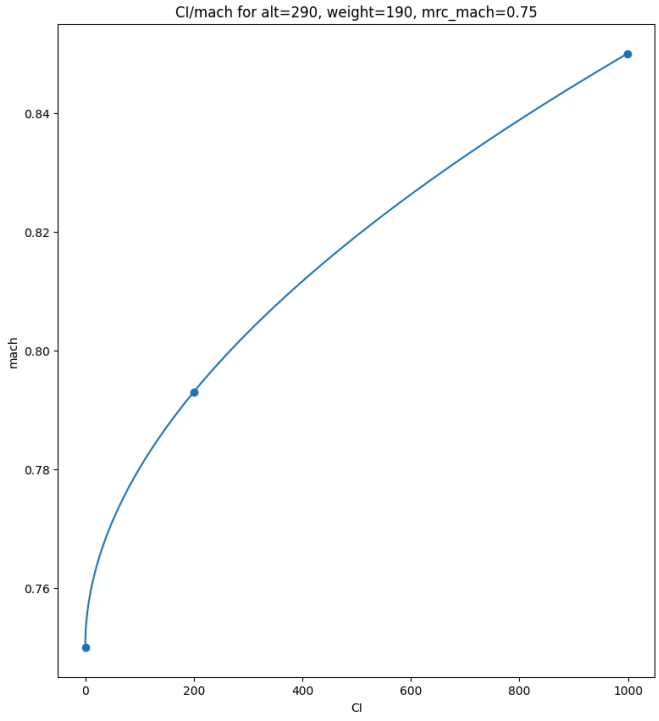




Sanity check: 0.711 MRC and 0.77 LRC

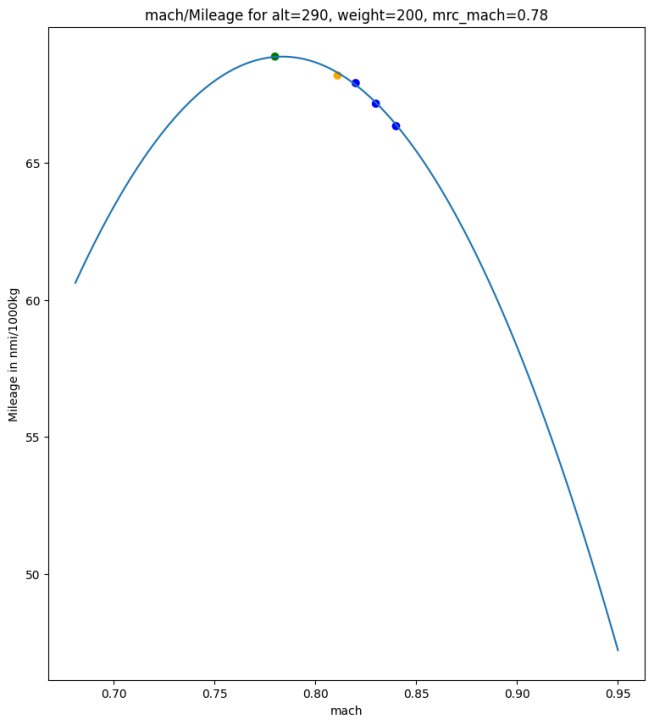
Found: 0.75 MRC and 0.793 LRC

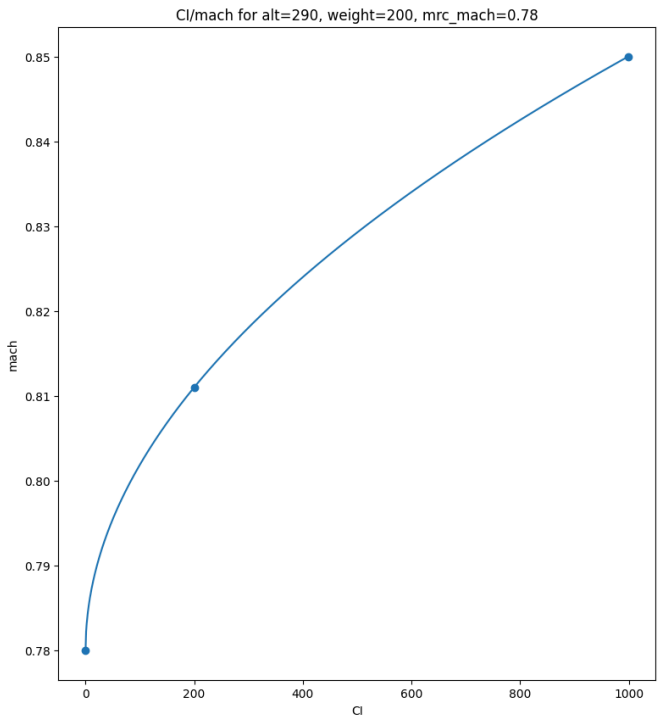




Sanity check: 0.75 MRC and 0.793 LRC

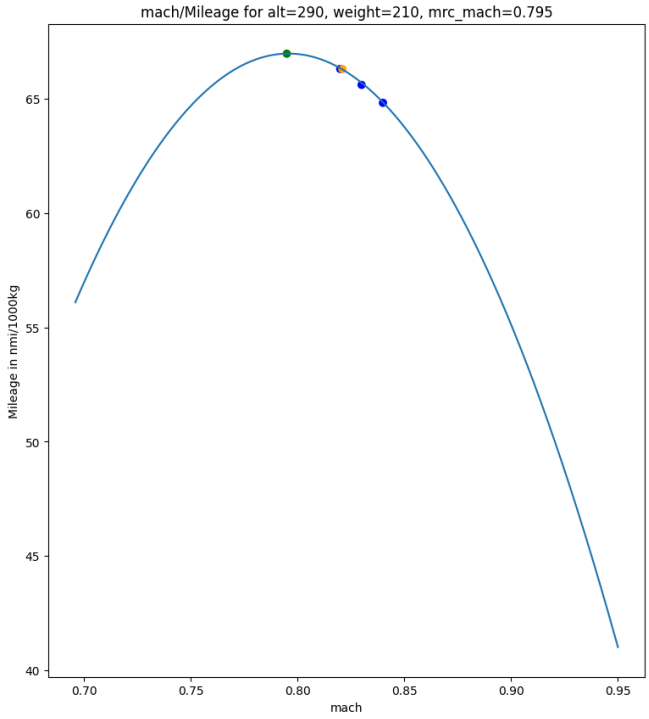
Found: 0.78 MRC and 0.811 LRC

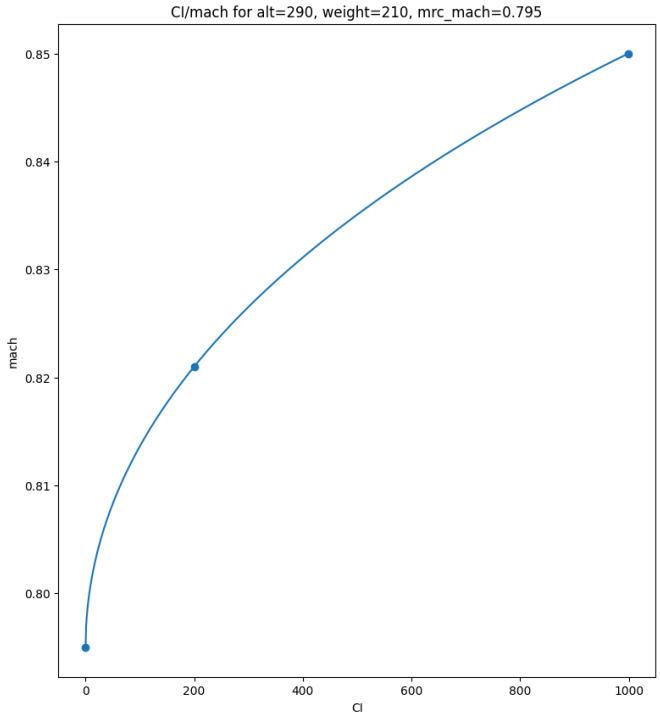




Sanity check: 0.78 MRC and 0.811 LRC

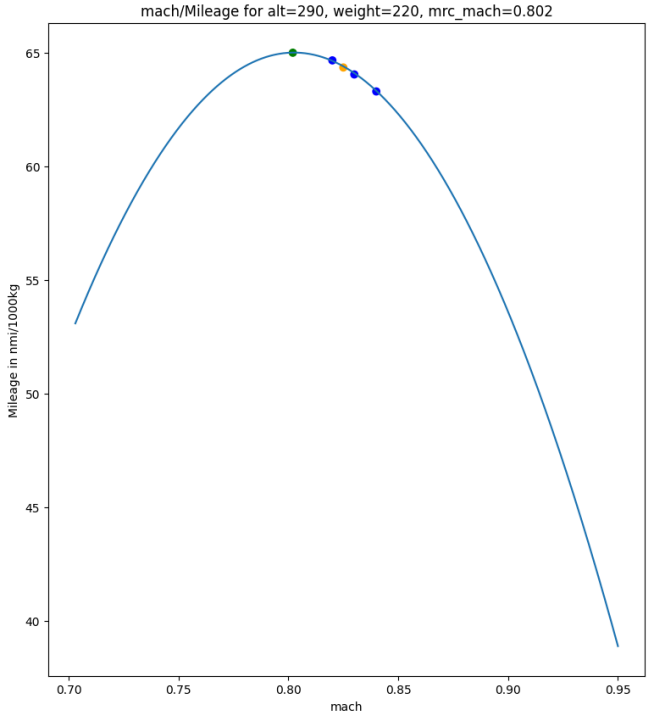
Found: 0.795 MRC and 0.821 LRC

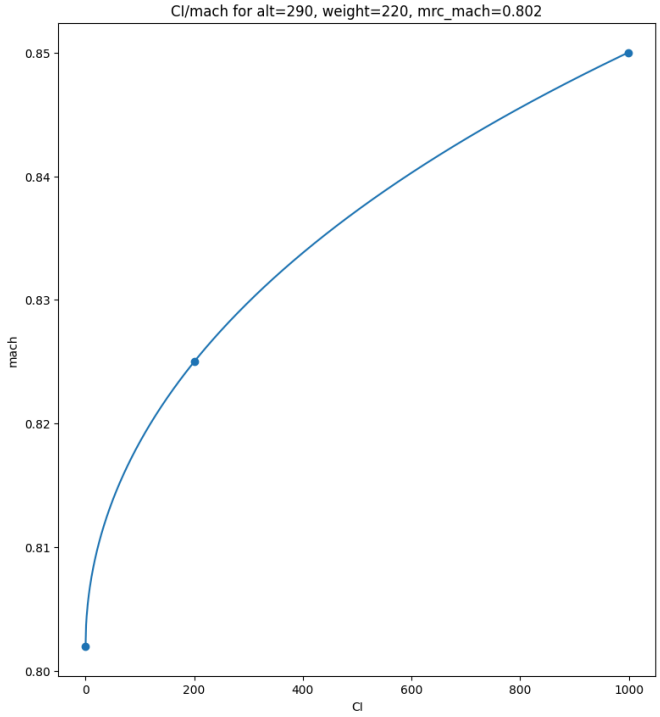




Sanity check: 0.795 MRC and 0.821 LRC

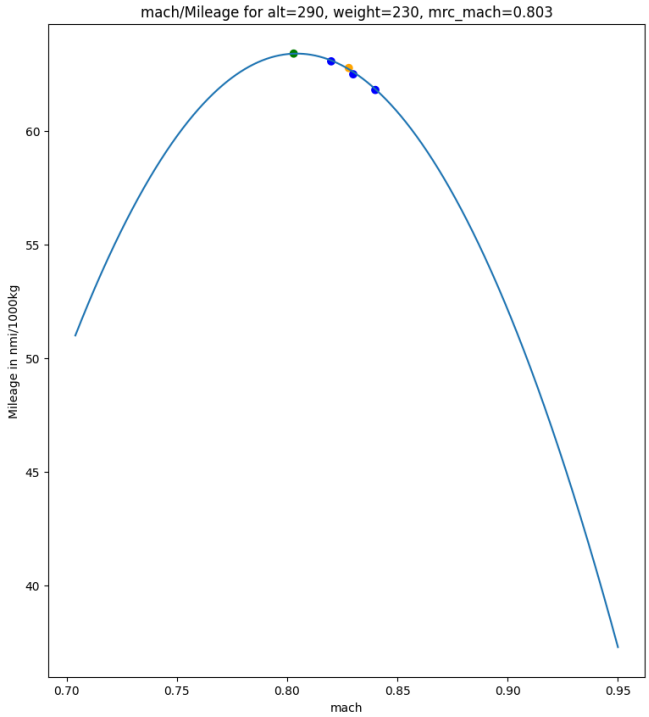
Found: 0.802 MRC and 0.825 LRC

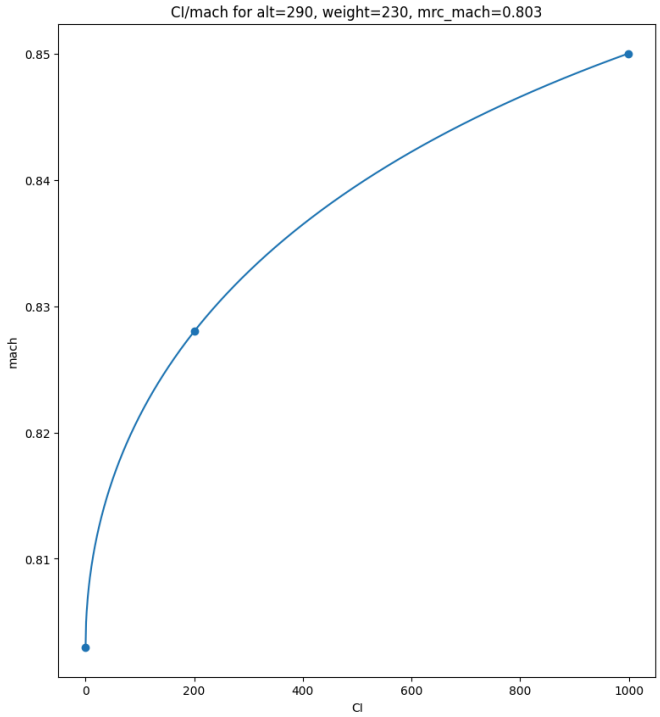




Sanity check: 0.802 MRC and 0.825 LRC

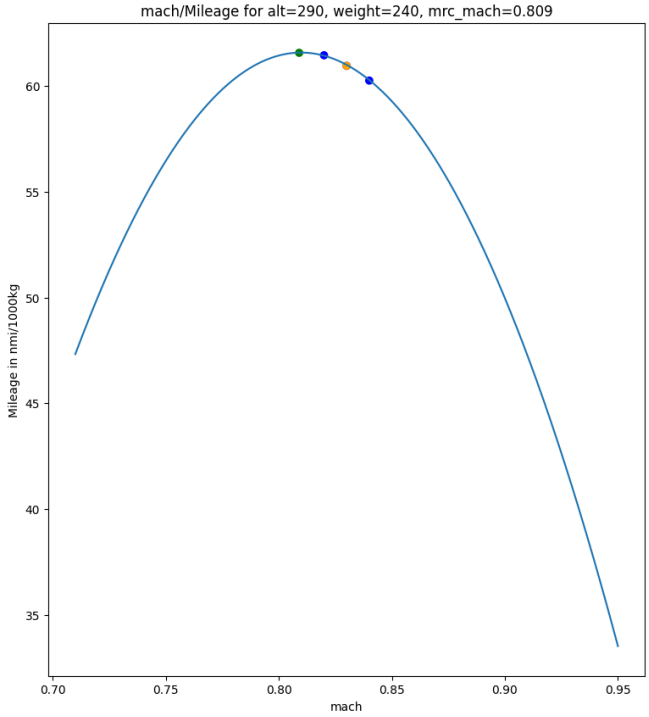
Found: 0.803 MRC and 0.828 LRC

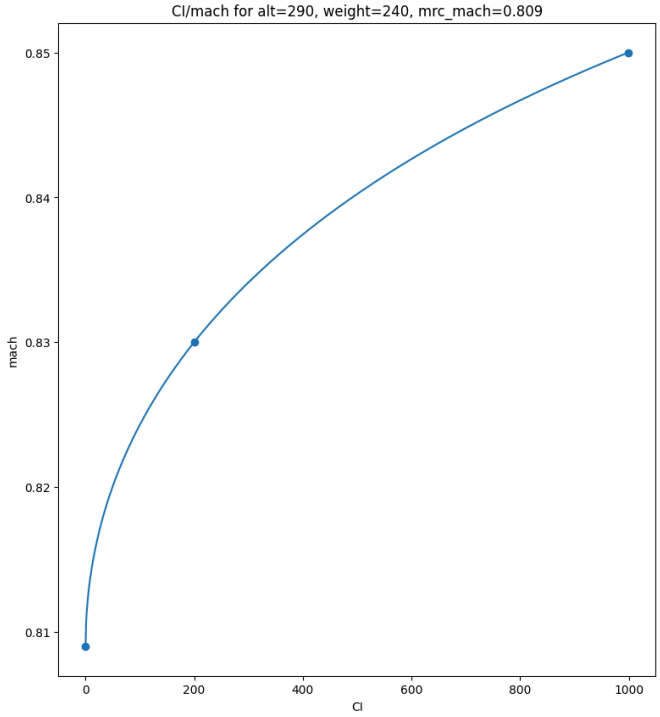




Sanity check: 0.803 MRC and 0.828 LRC

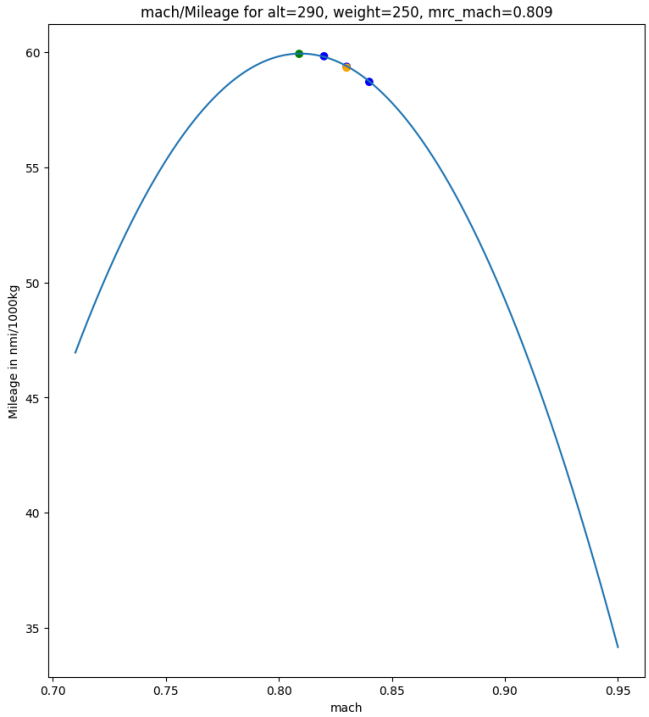
Found: 0.809 MRC and 0.83 LRC

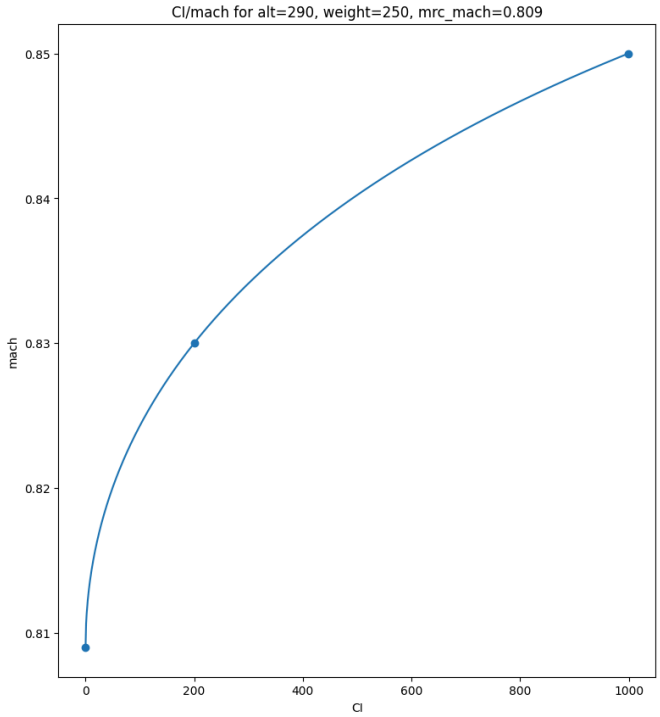




Sanity check: 0.809 MRC and 0.83 LRC

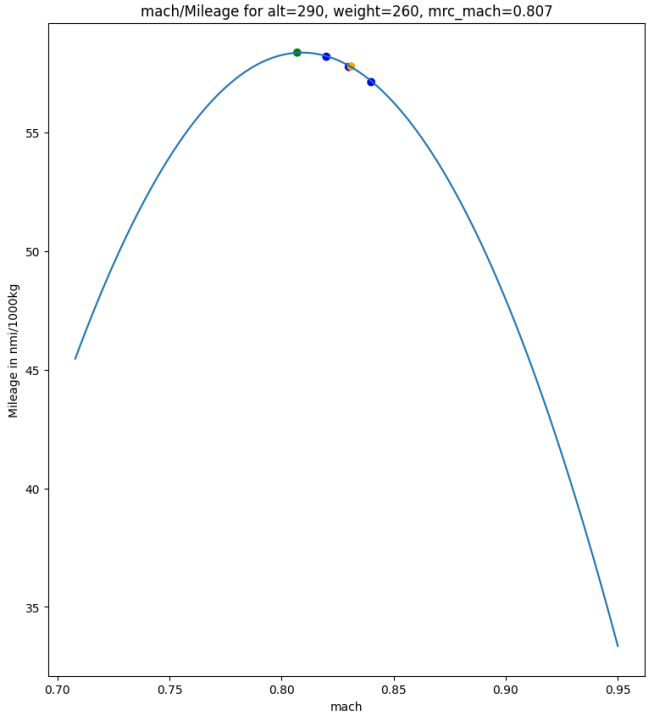
Found: 0.809 MRC and 0.83 LRC

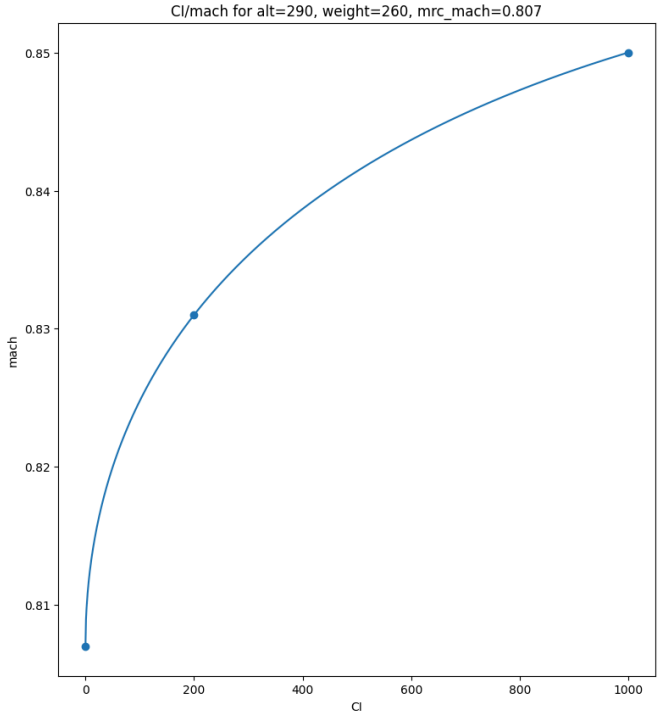




Sanity check: 0.809 MRC and 0.83 LRC

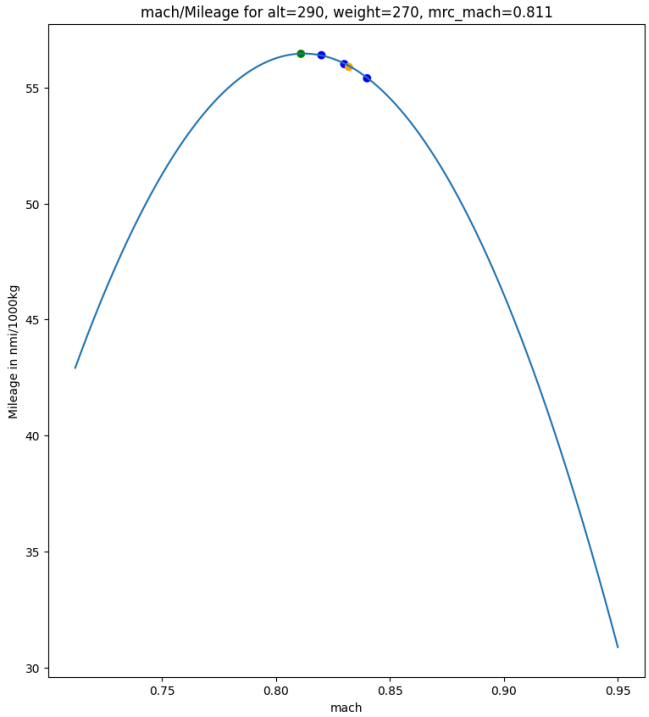
Found: 0.807 MRC and 0.831 LRC

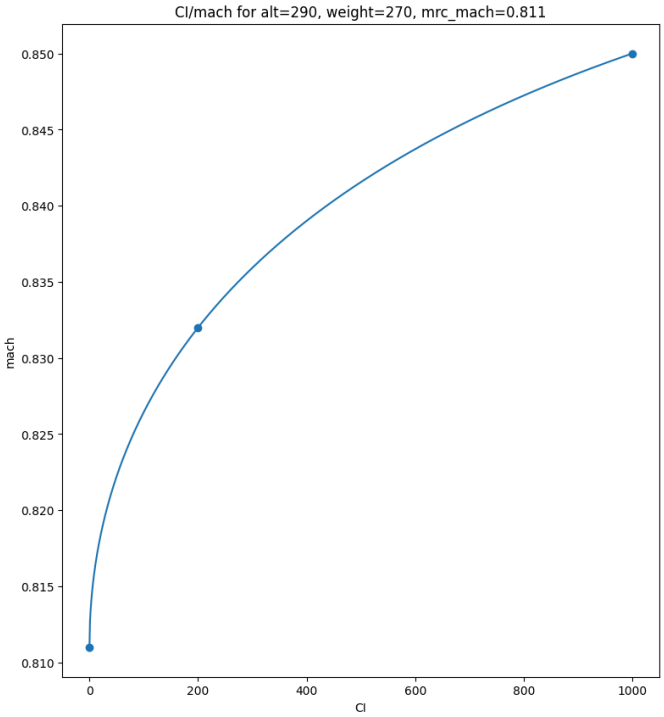




Sanity check: 0.807 MRC and 0.831 LRC

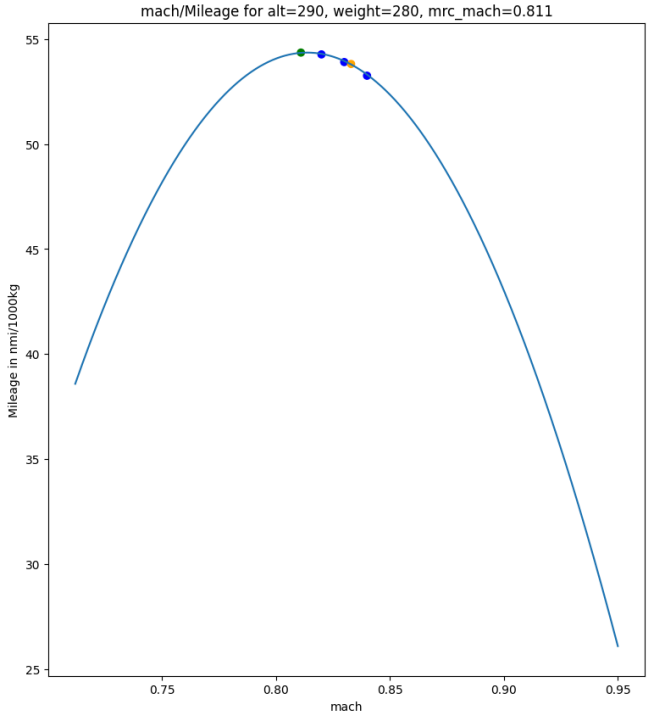
Found: 0.811 MRC and 0.832 LRC

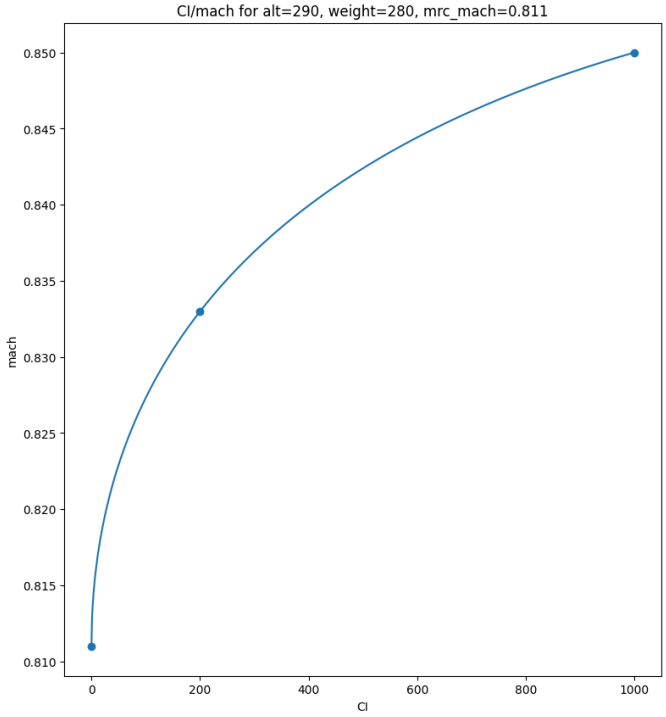




Sanity check: 0.811 MRC and 0.832 LRC

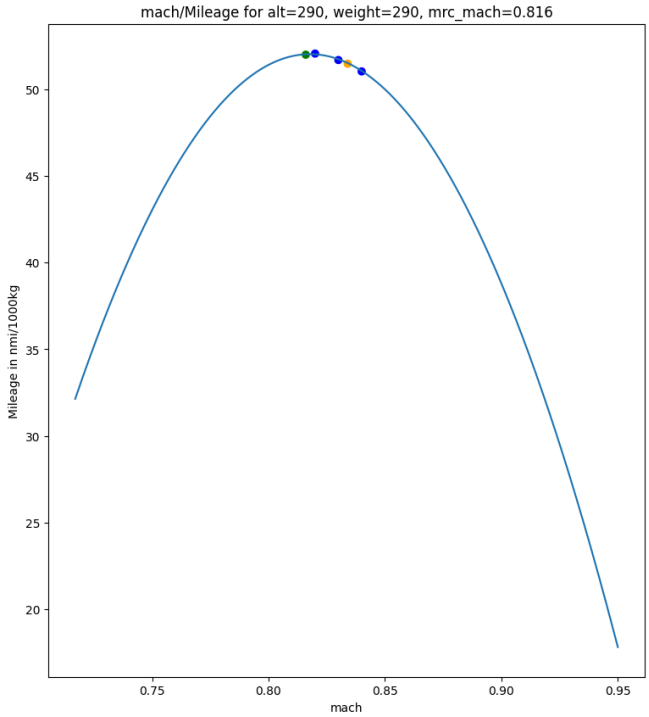
Found: 0.811 MRC and 0.833 LRC

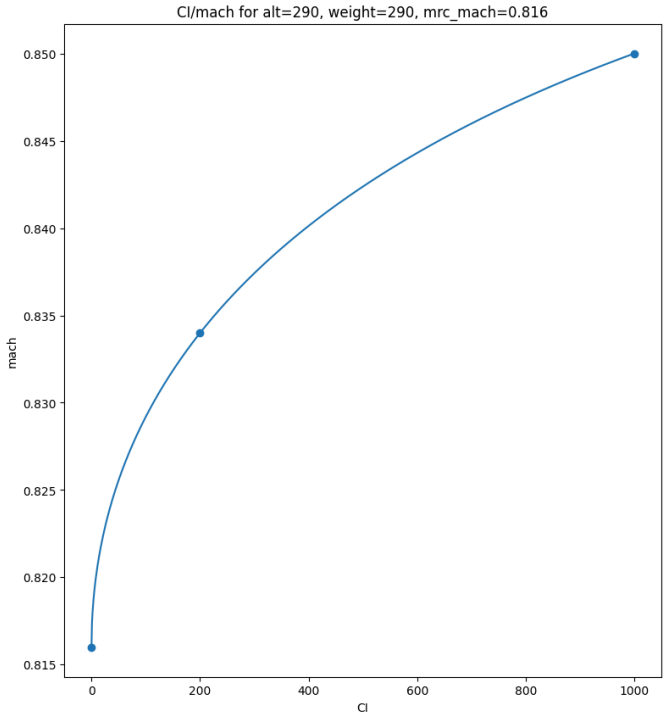




Sanity check: 0.811 MRC and 0.833 LRC

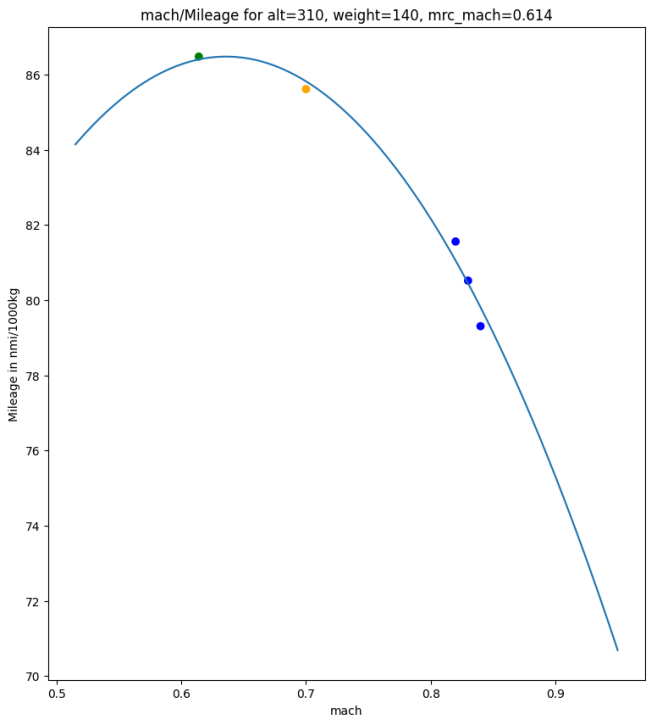
Found: 0.816 MRC and 0.834 LRC

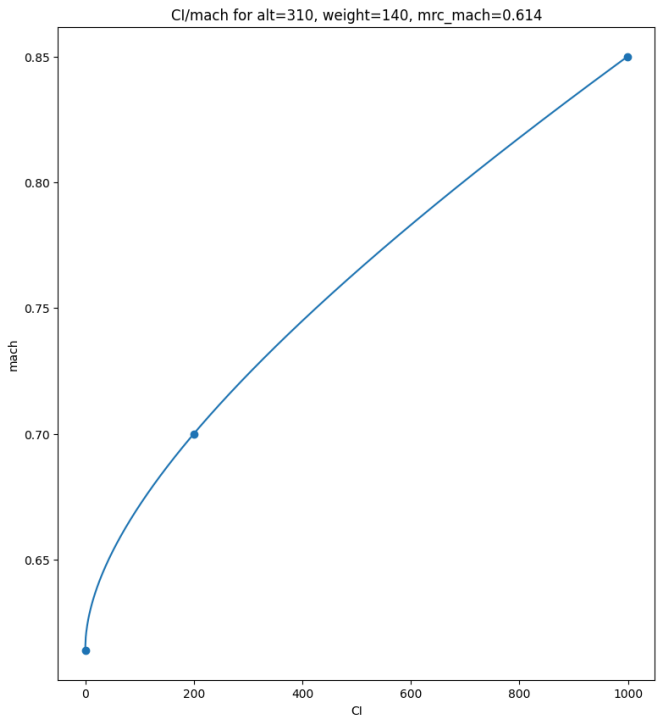




Sanity check: 0.816 MRC and 0.834 LRC

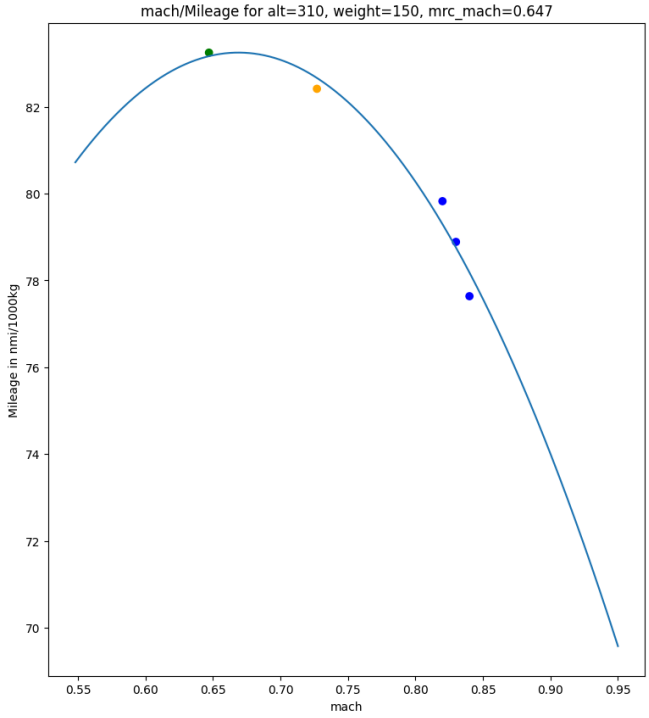
Found: 0.614 MRC and 0.7 LRC

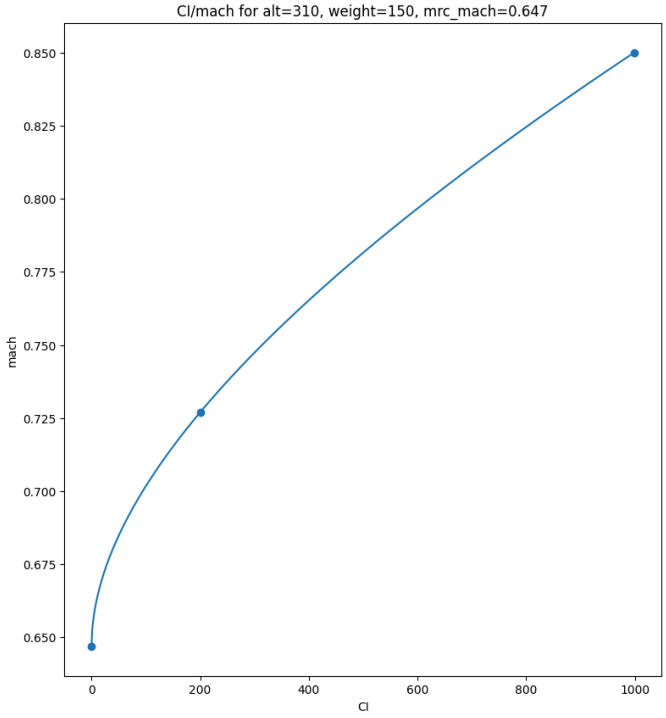




Sanity check: 0.614 MRC and 0.7 LRC

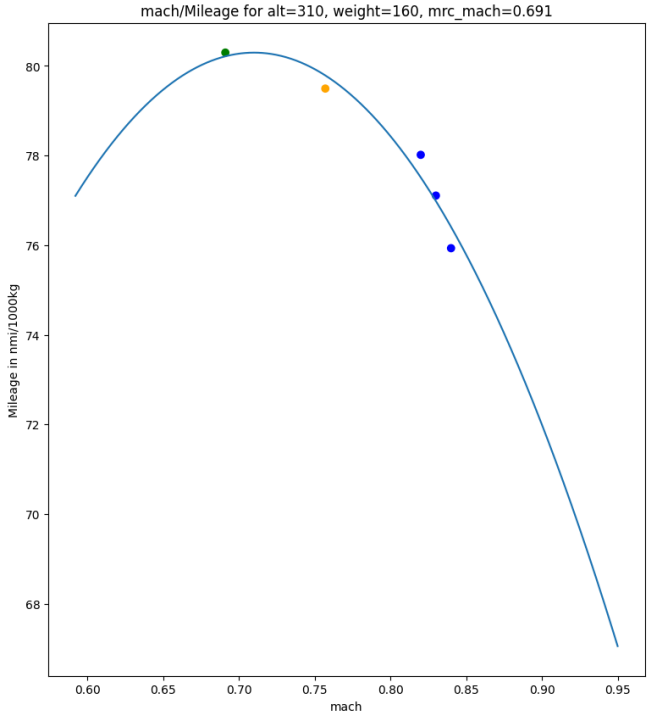
Found: 0.647 MRC and 0.727 LRC

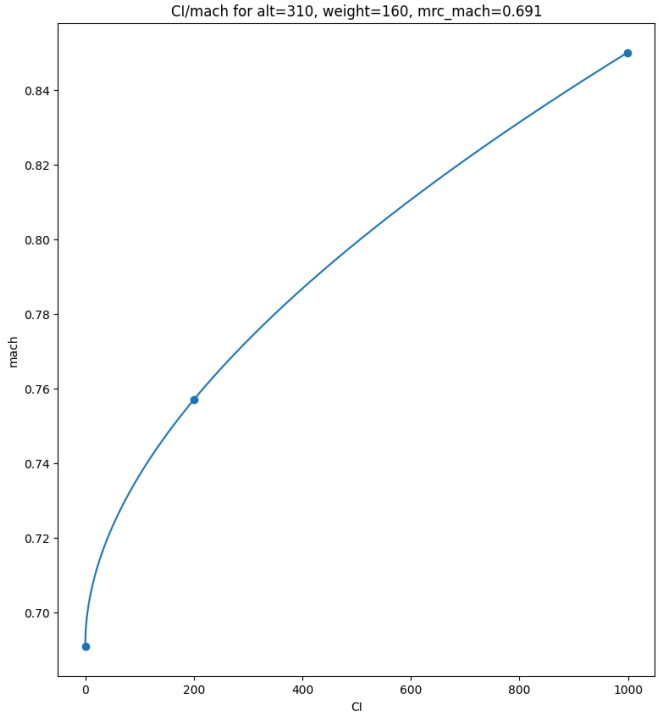




Sanity check: 0.647 MRC and 0.727 LRC

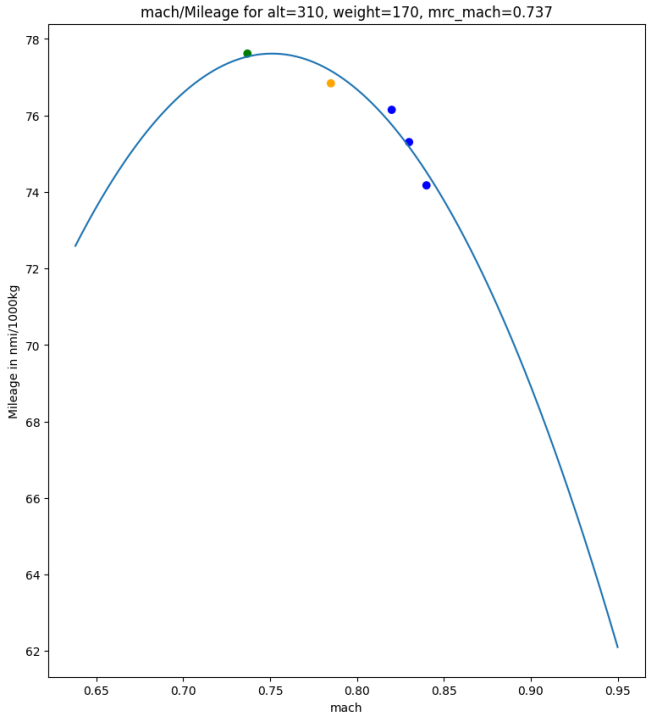
Found: 0.691 MRC and 0.757 LRC

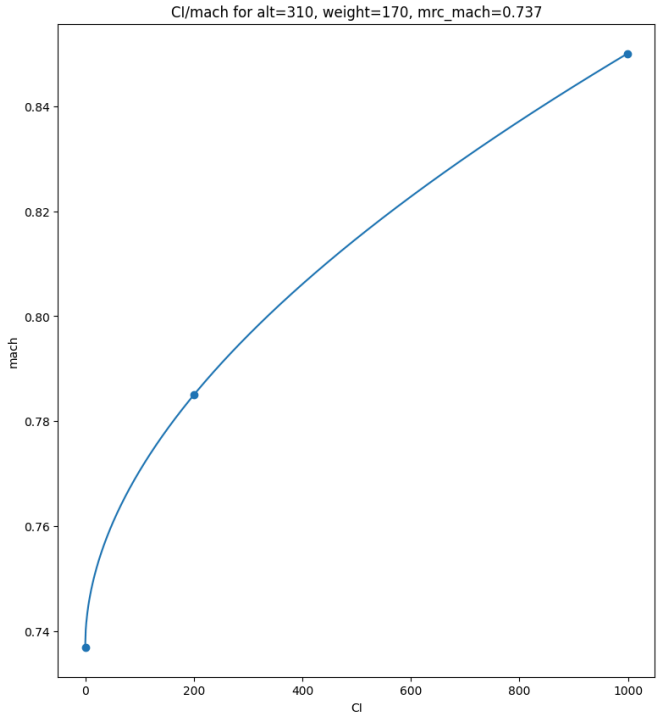




Sanity check: 0.691 MRC and 0.757 LRC

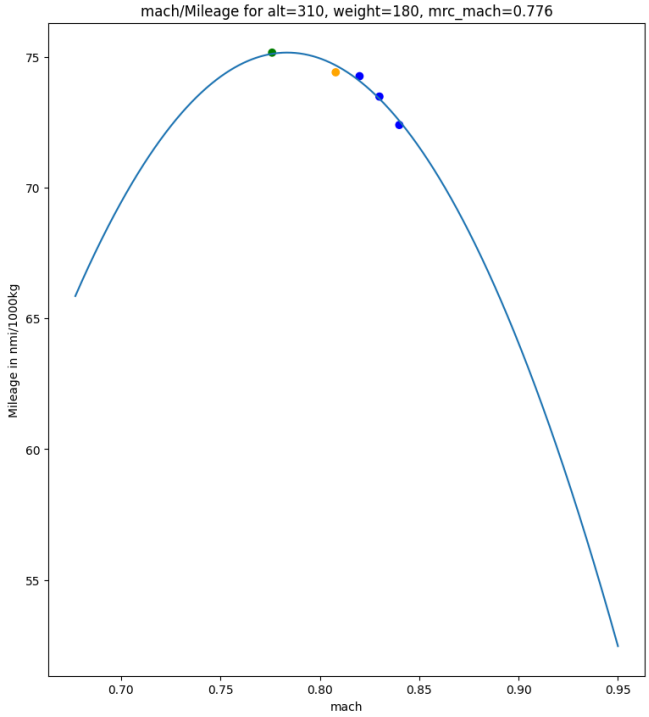
Found: 0.737 MRC and 0.785 LRC

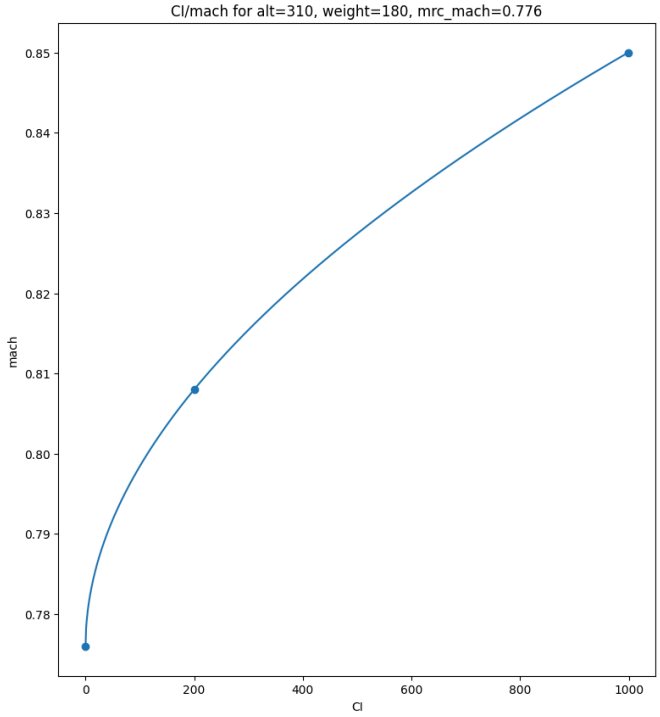




Sanity check: 0.737 MRC and 0.785 LRC

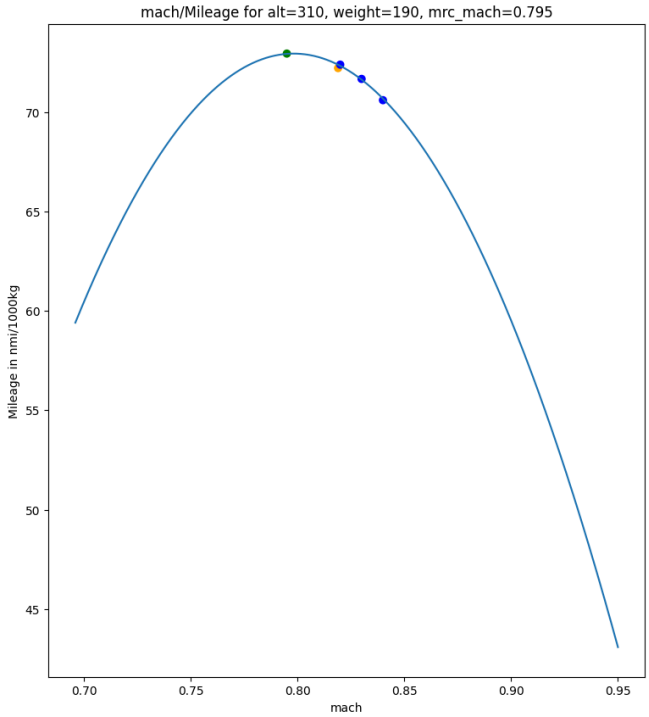
Found: 0.776 MRC and 0.808 LRC

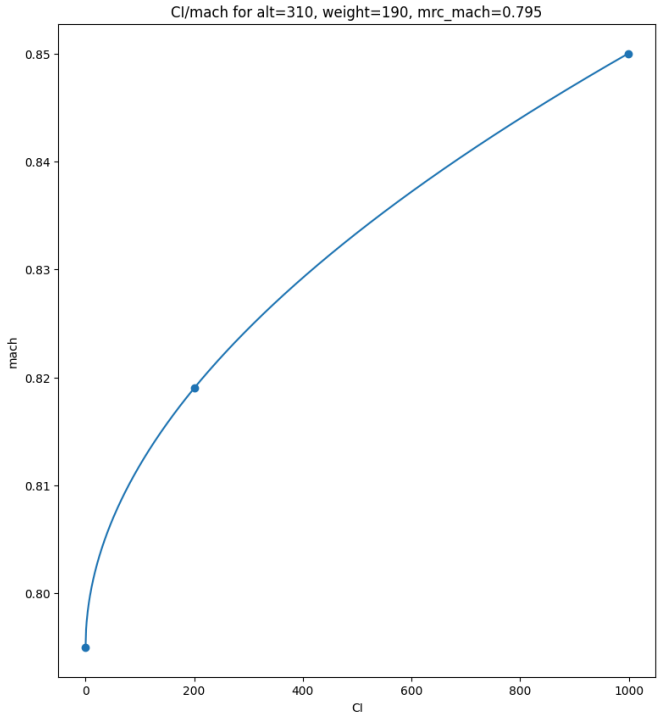




Sanity check: 0.776 MRC and 0.808 LRC

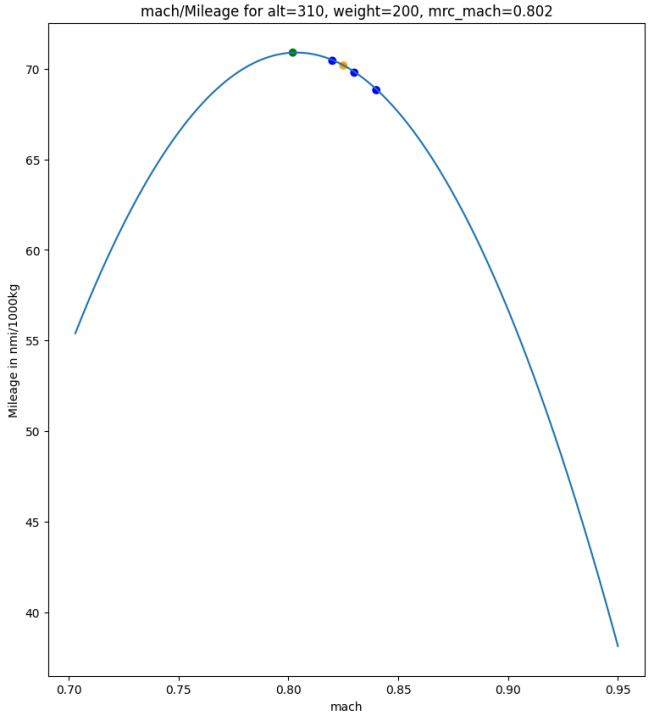
Found: 0.795 MRC and 0.819 LRC

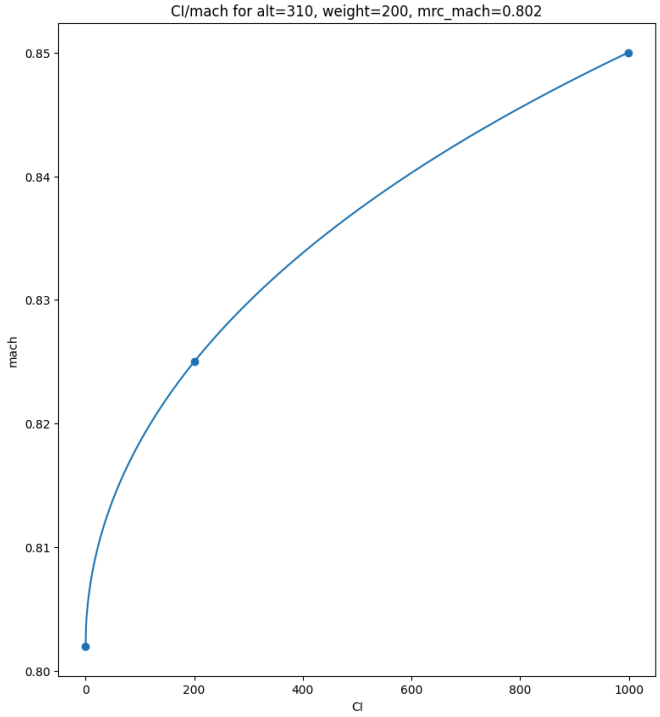




Sanity check: 0.795 MRC and 0.819 LRC

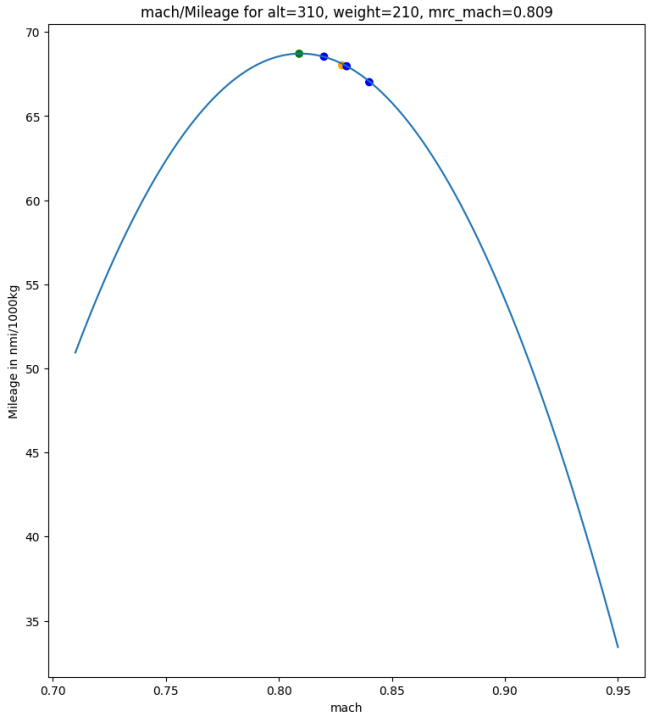
Found: 0.802 MRC and 0.825 LRC

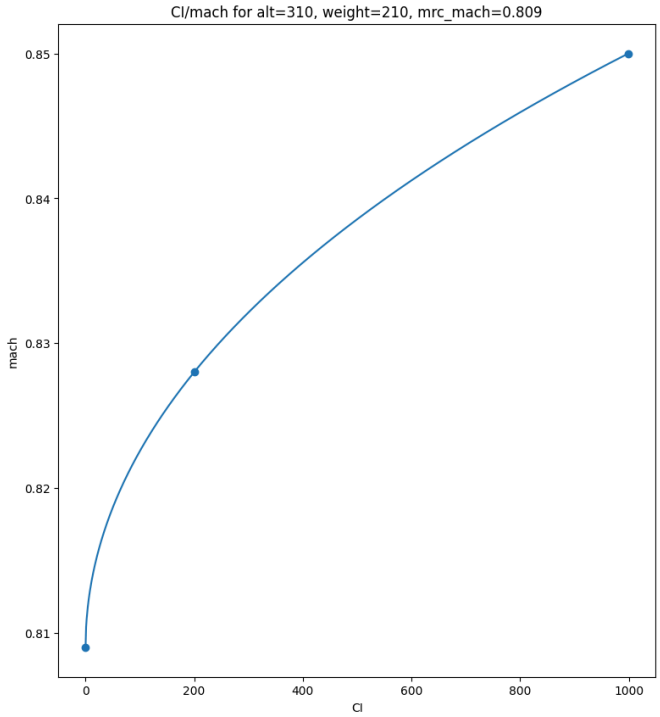




Sanity check: 0.802 MRC and 0.825 LRC

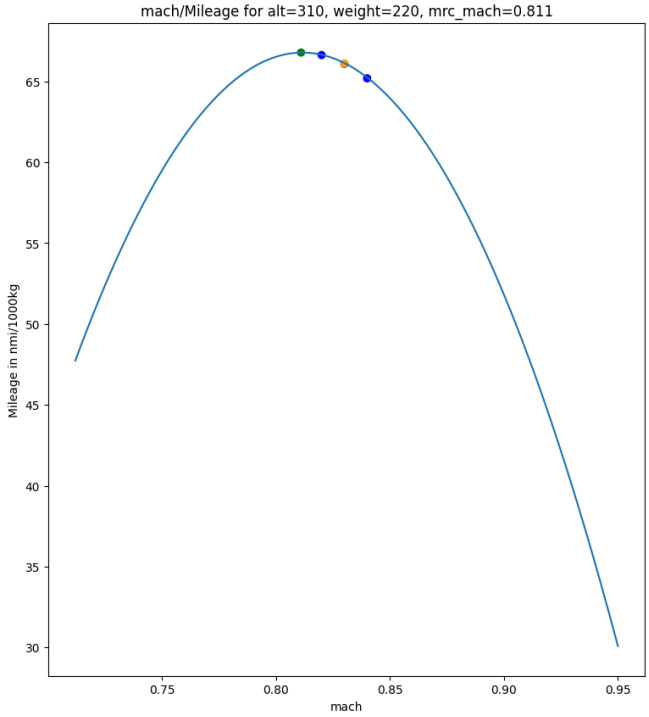
Found: 0.809 MRC and 0.828 LRC

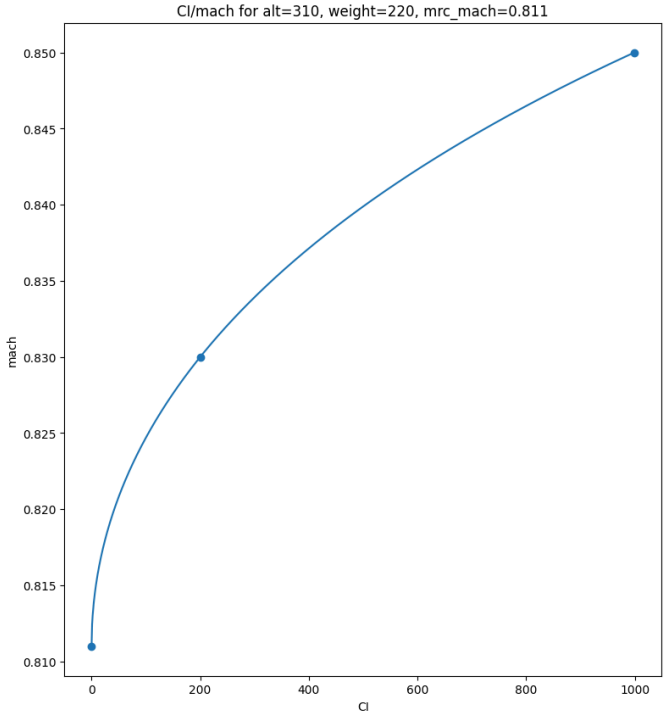




Sanity check: 0.809 MRC and 0.828 LRC

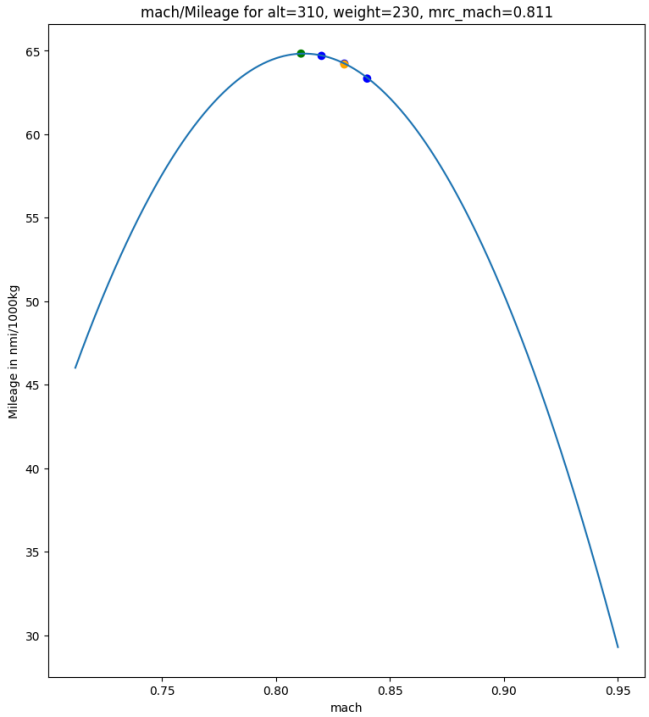
Found: 0.811 MRC and 0.83 LRC

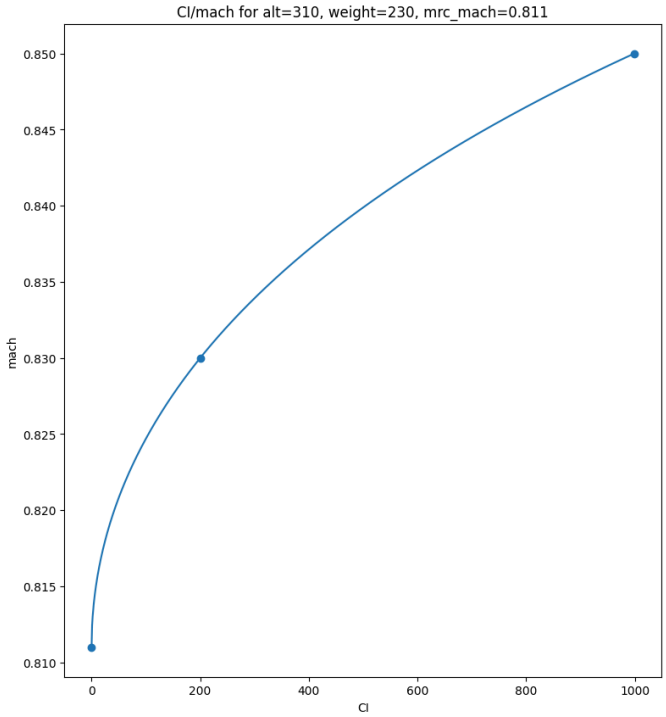




Sanity check: 0.811 MRC and 0.83 LRC

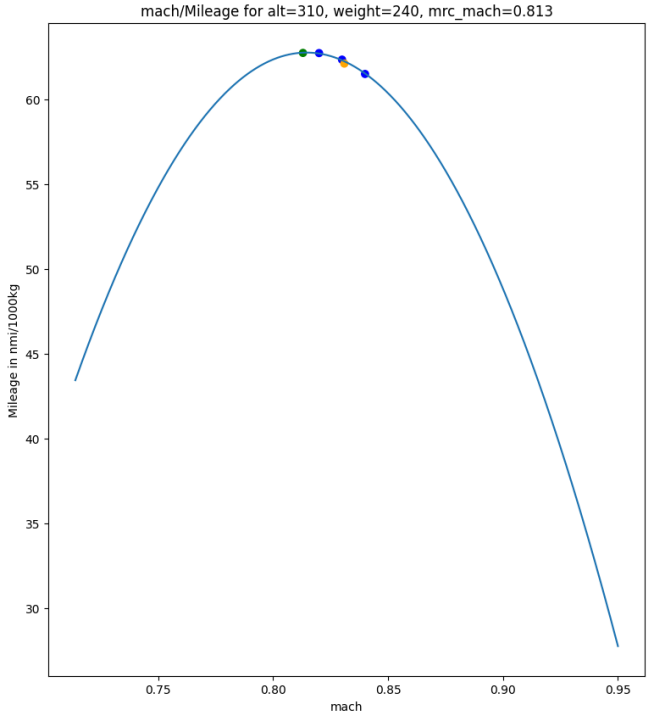
Found: 0.811 MRC and 0.83 LRC

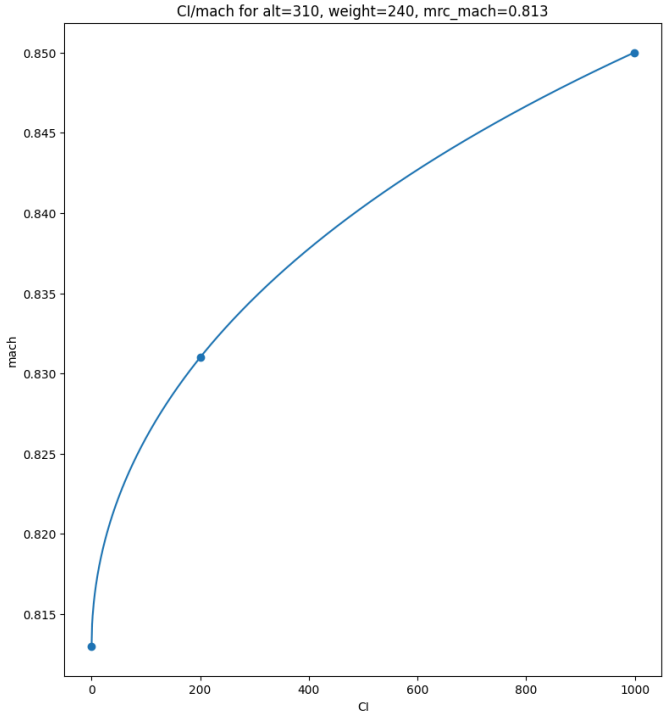




Sanity check: 0.811 MRC and 0.83 LRC

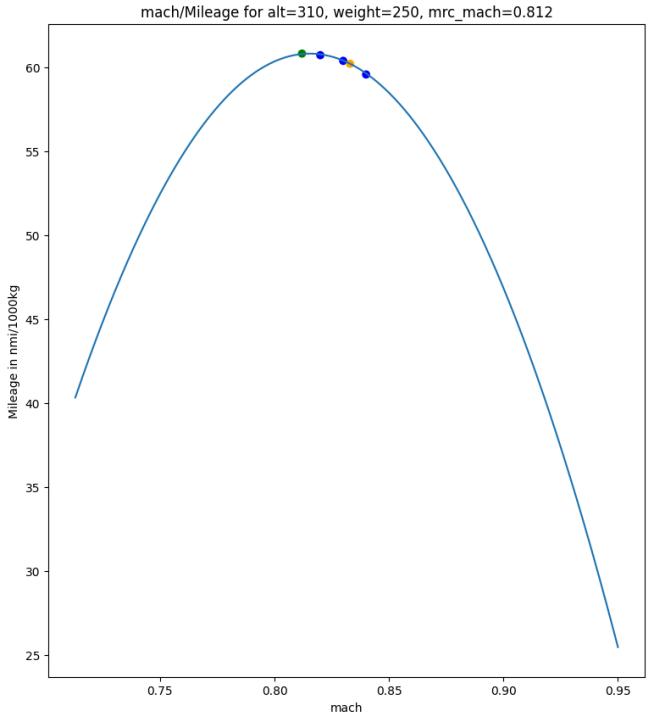
Found: 0.813 MRC and 0.831 LRC

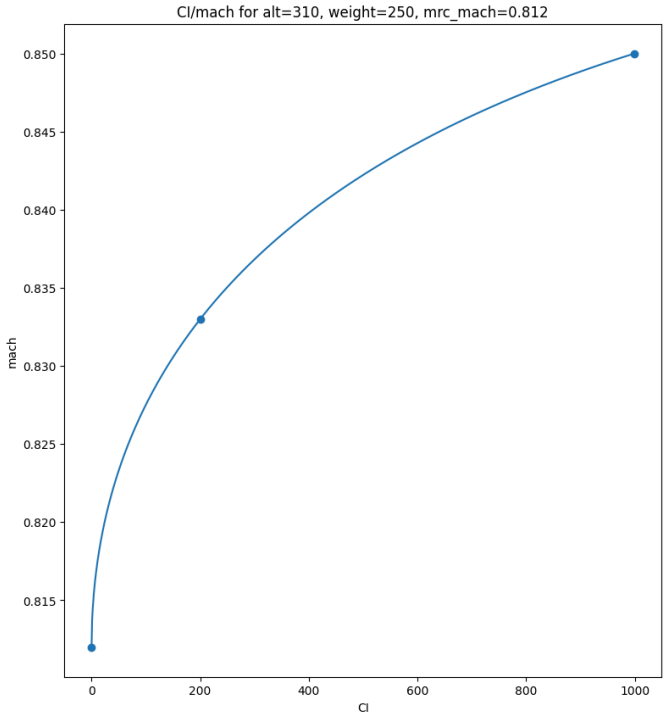




Sanity check: 0.813 MRC and 0.831 LRC

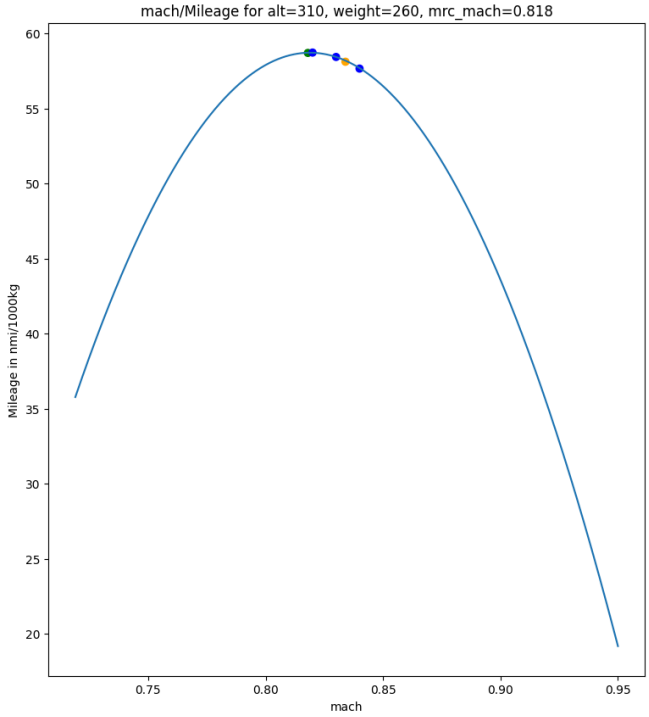
Found: 0.812 MRC and 0.833 LRC

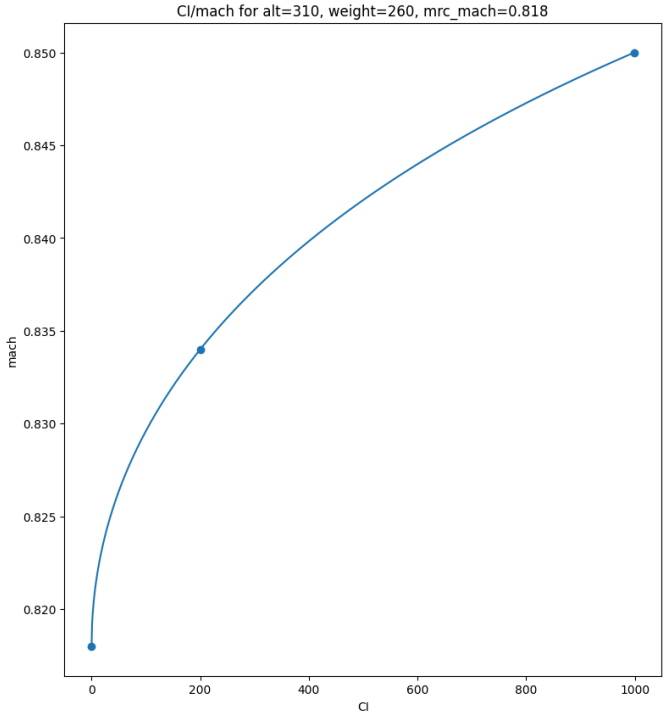




Sanity check: 0.812 MRC and 0.833 LRC

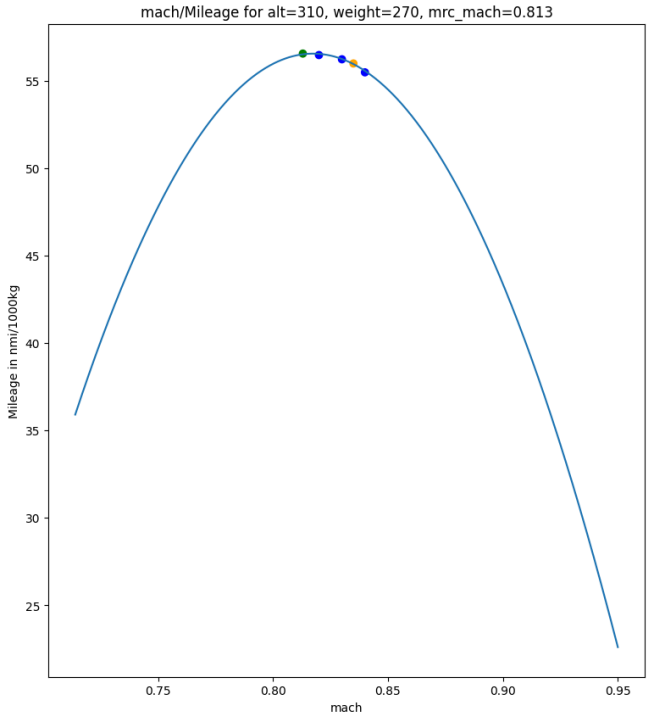
Found: 0.818 MRC and 0.834 LRC

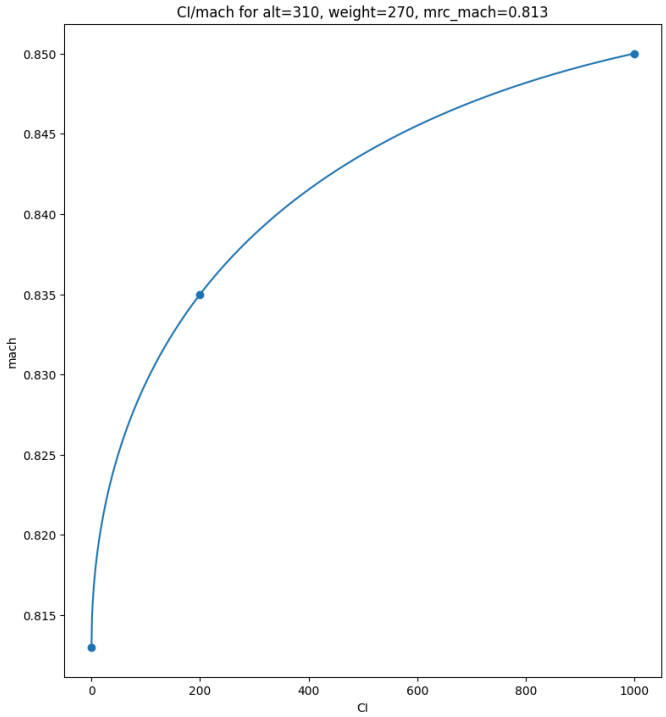




Sanity check: 0.818 MRC and 0.834 LRC

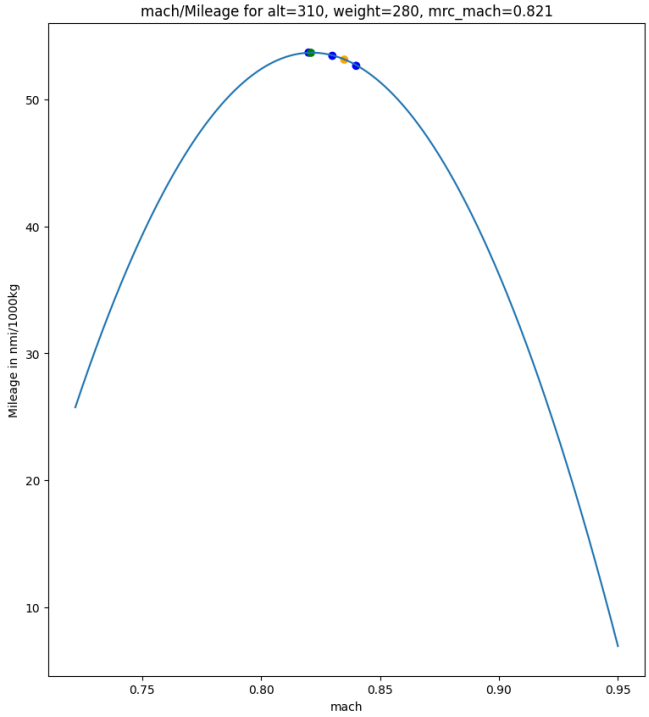
Found: 0.813 MRC and 0.835 LRC

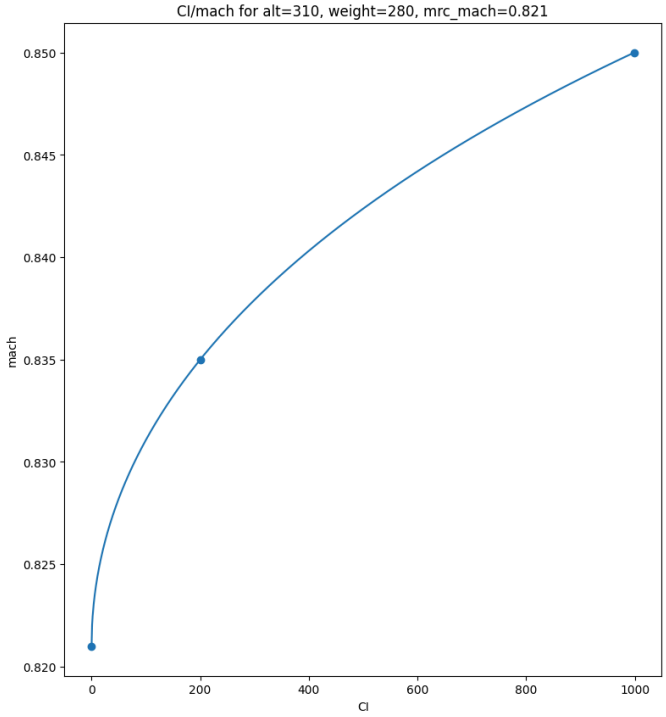




Sanity check: 0.813 MRC and 0.835 LRC

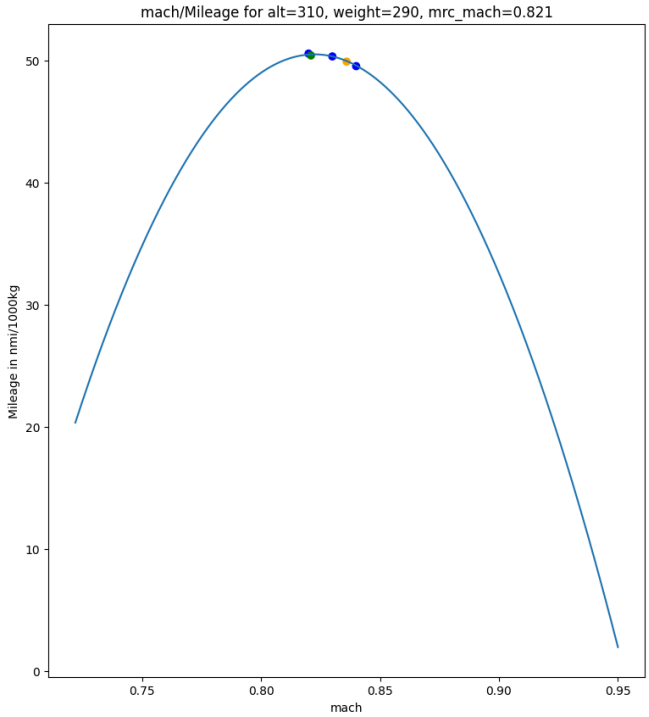
Found: 0.821 MRC and 0.835 LRC

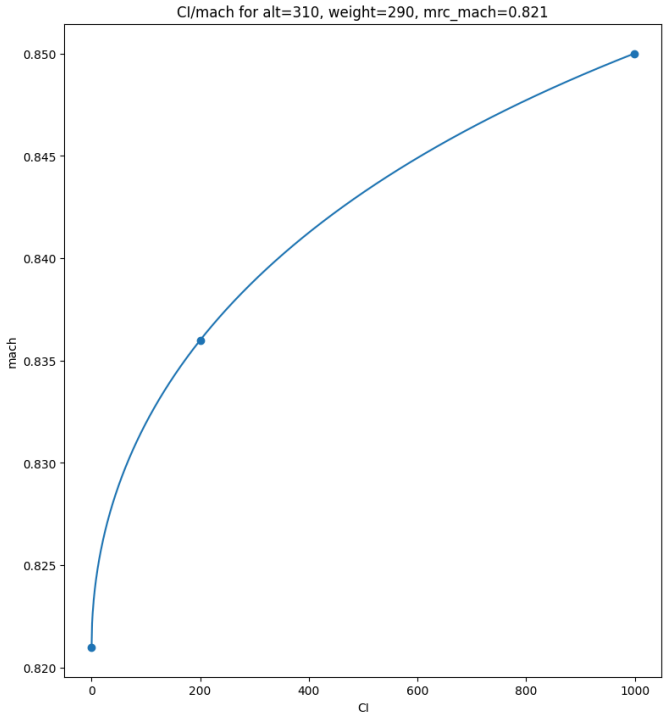




Sanity check: 0.821 MRC and 0.835 LRC

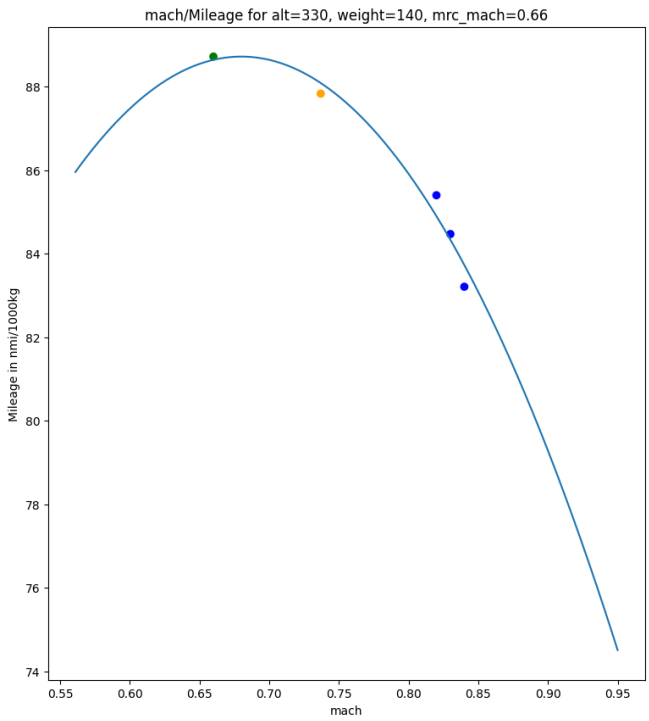
Found: 0.821 MRC and 0.836 LRC

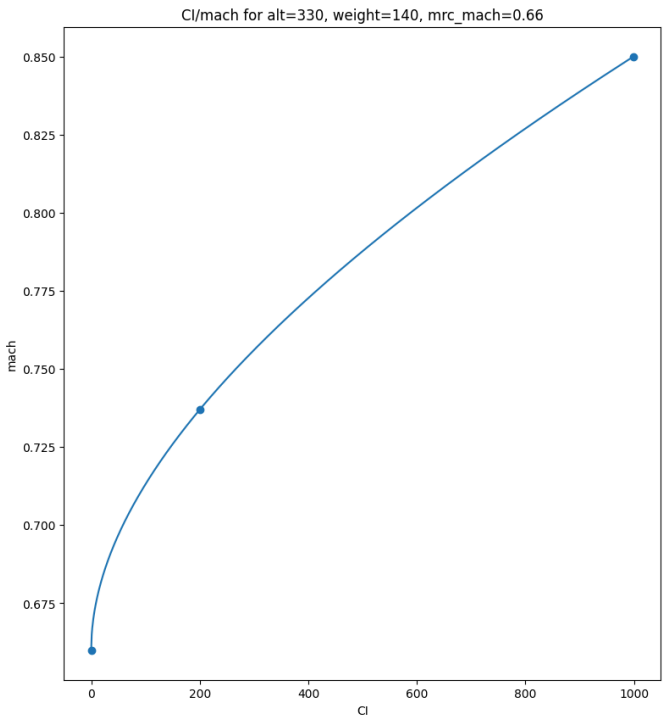




Sanity check: 0.821 MRC and 0.836 LRC

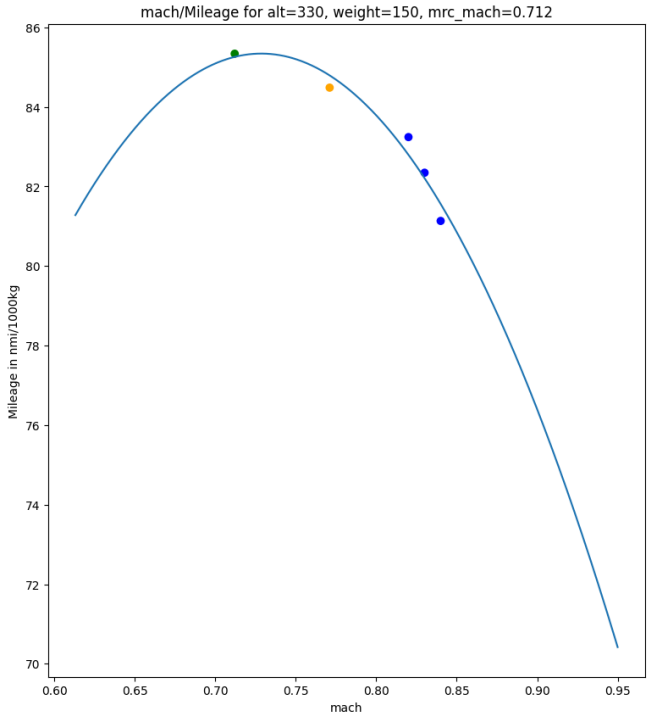
Found: 0.66 MRC and 0.737 LRC

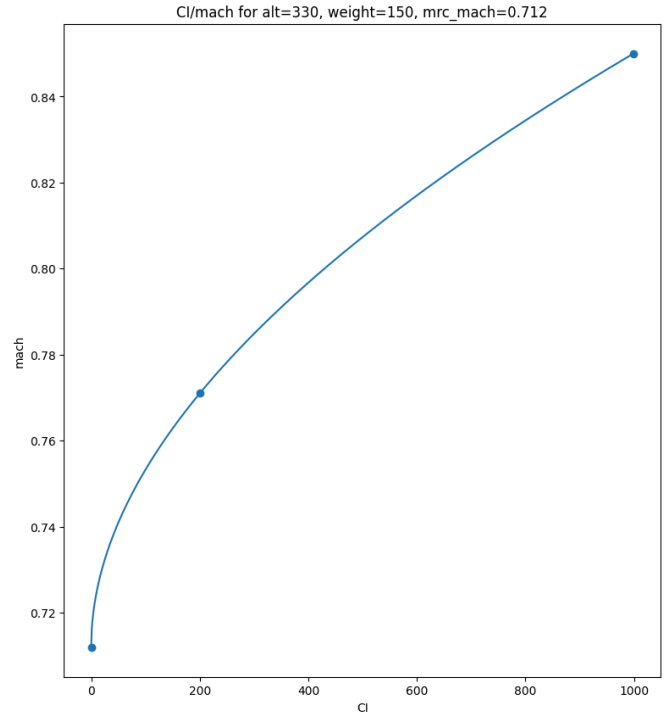




Sanity check: 0.66 MRC and 0.737 LRC

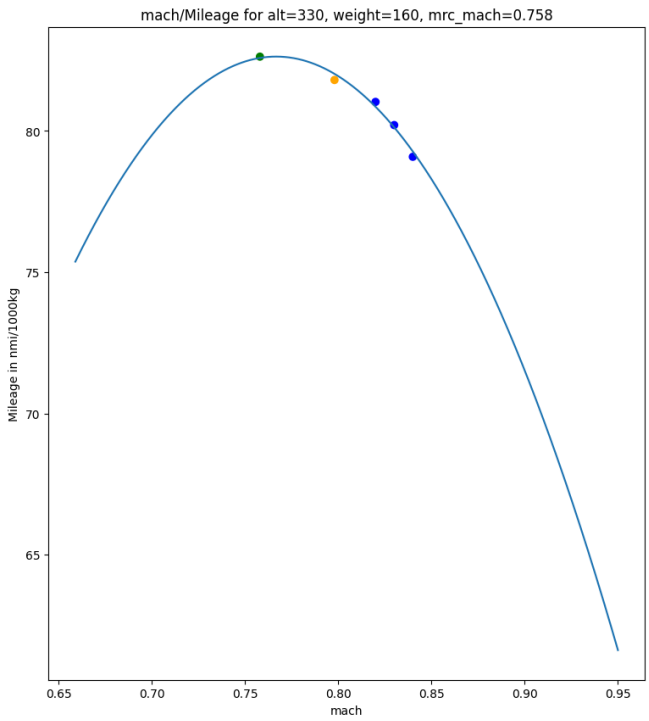
Found: 0.712 MRC and 0.771 LRC

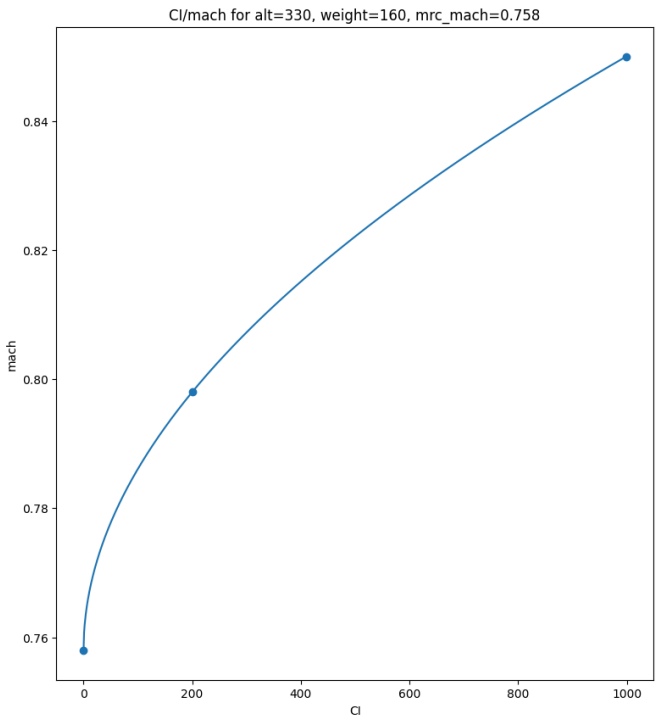




Sanity check: 0.712 MRC and 0.771 LRC

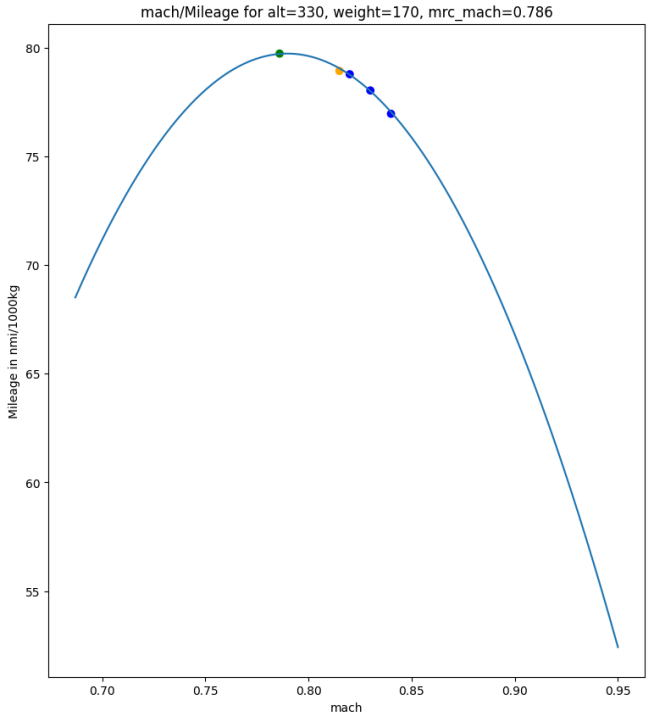
Found: 0.758 MRC and 0.798 LRC

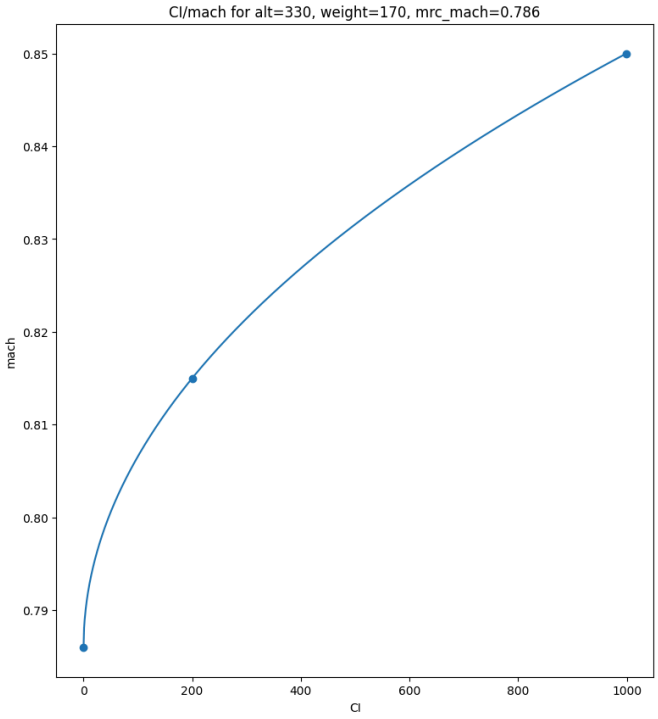




Sanity check: 0.758 MRC and 0.798 LRC

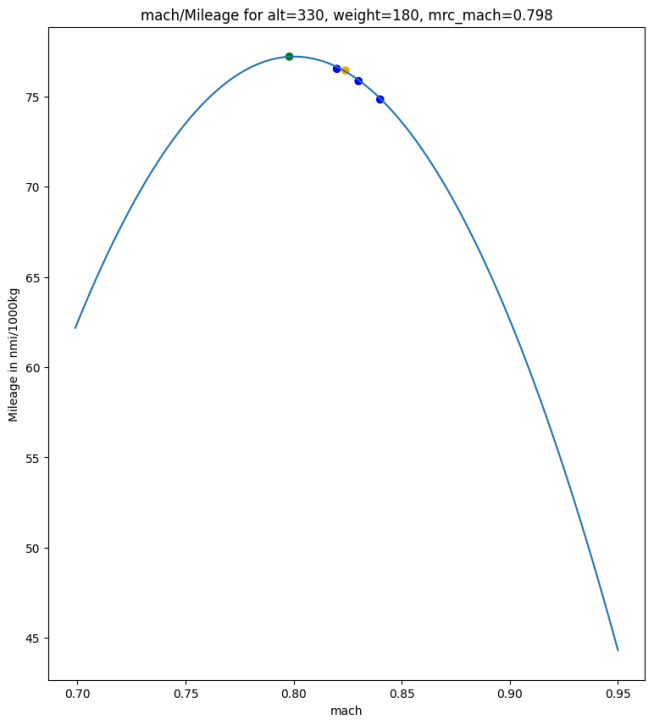
Found: 0.786 MRC and 0.815 LRC

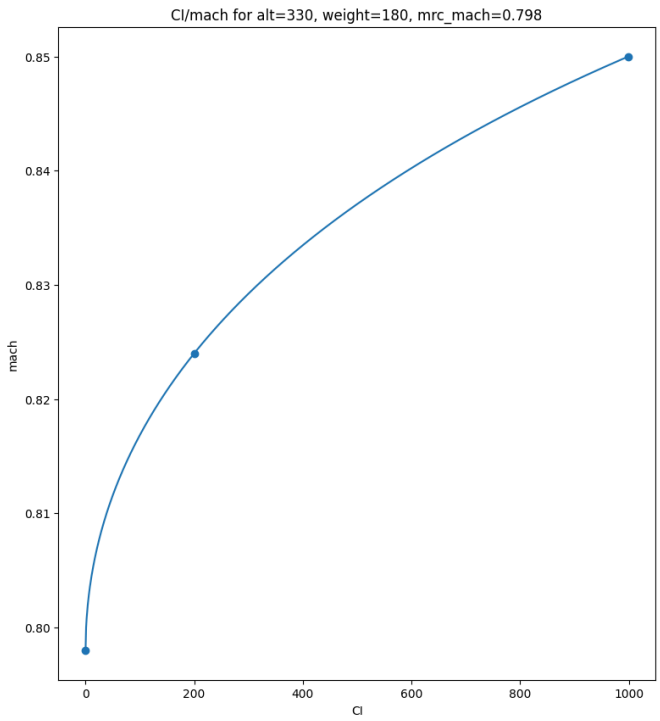




Sanity check: 0.786 MRC and 0.815 LRC

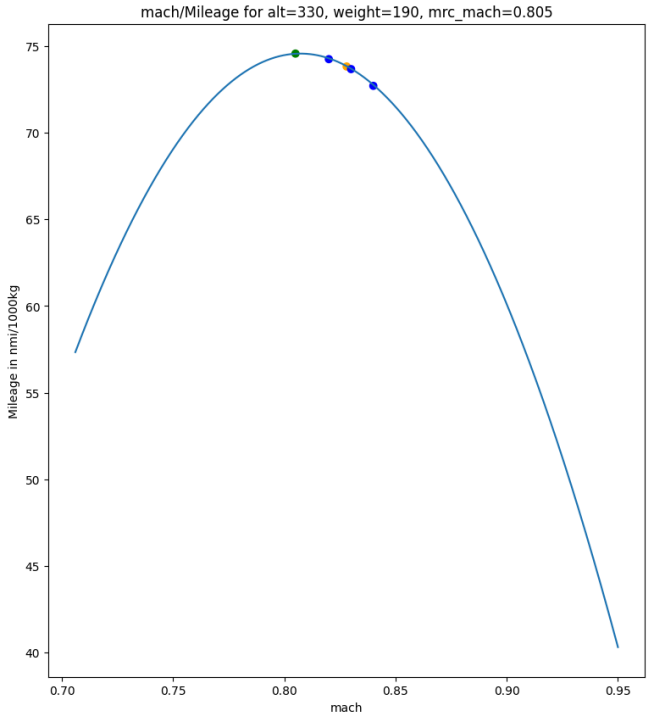
Found: 0.798 MRC and 0.824 LRC

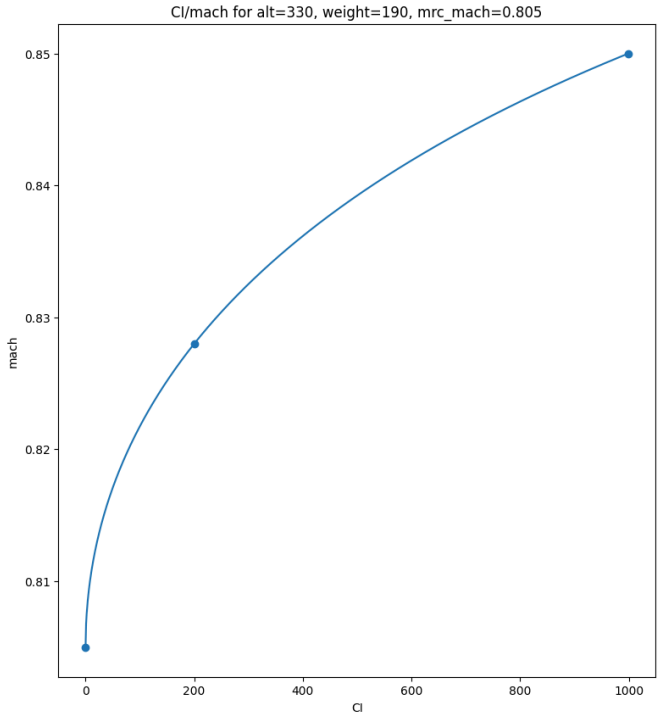




Sanity check: 0.798 MRC and 0.824 LRC

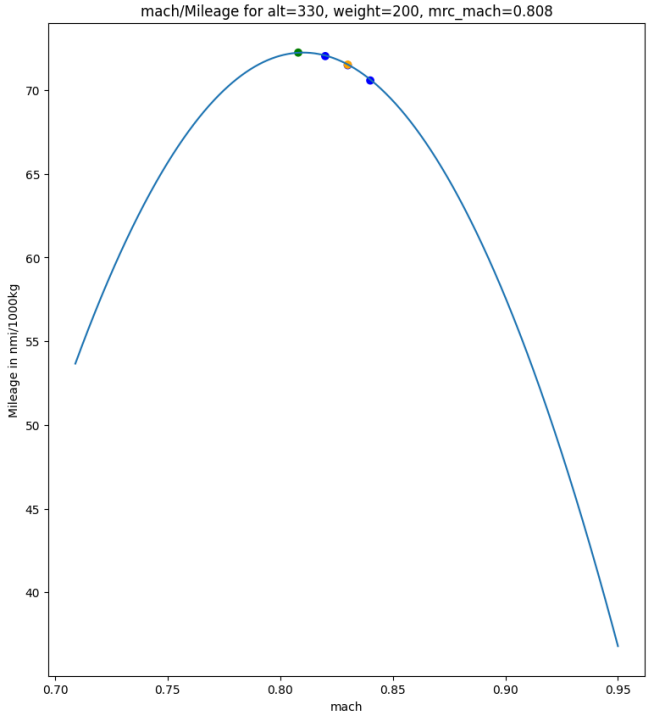
Found: 0.805 MRC and 0.828 LRC

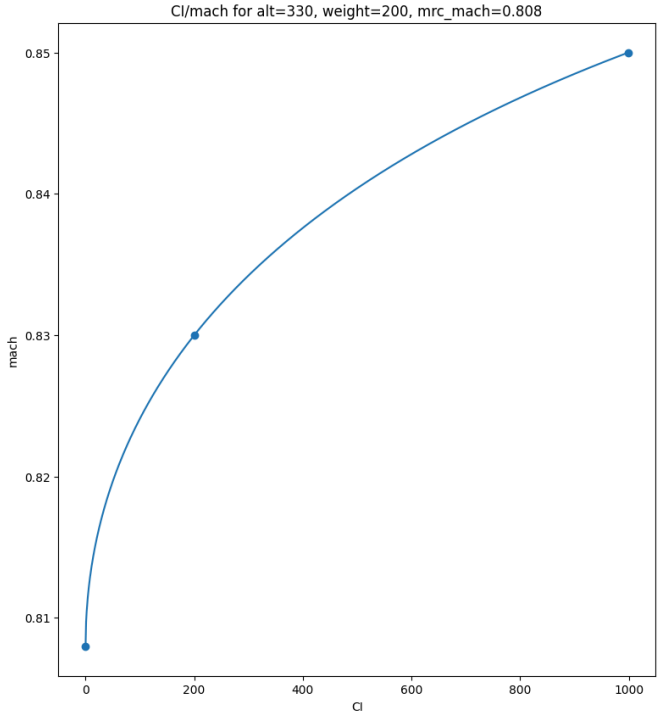




Sanity check: 0.805 MRC and 0.828 LRC

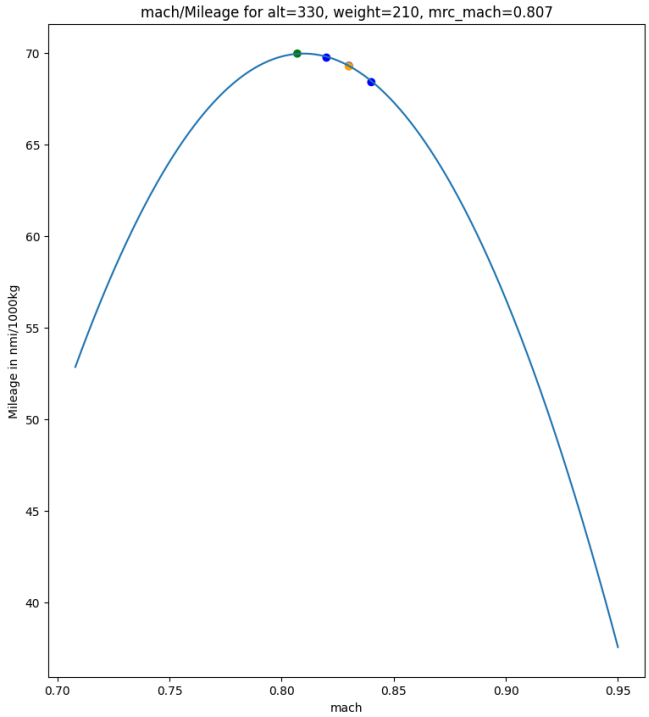
Found: 0.808 MRC and 0.83 LRC

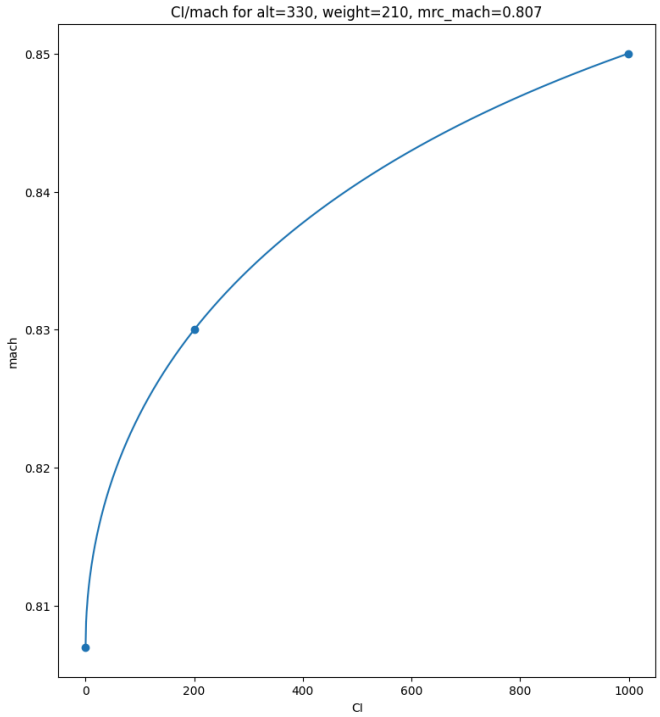




Sanity check: 0.808 MRC and 0.83 LRC

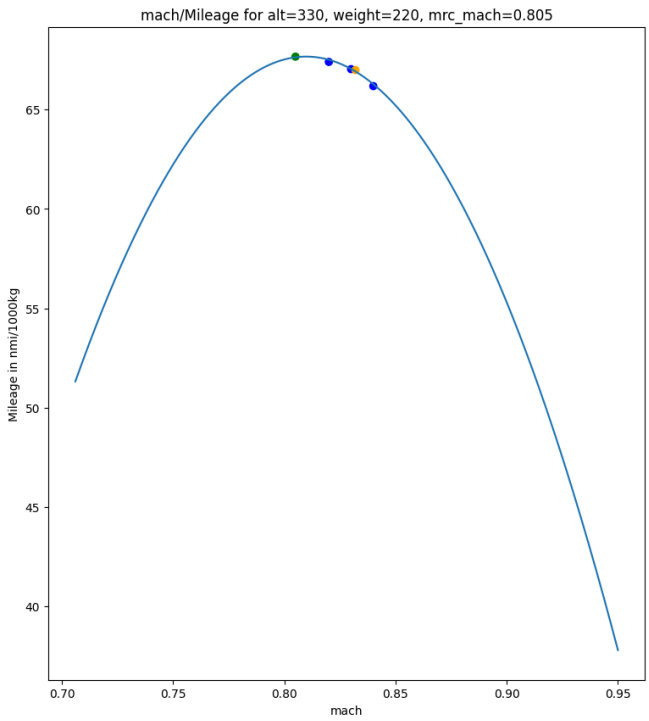
Found: 0.807 MRC and 0.83 LRC

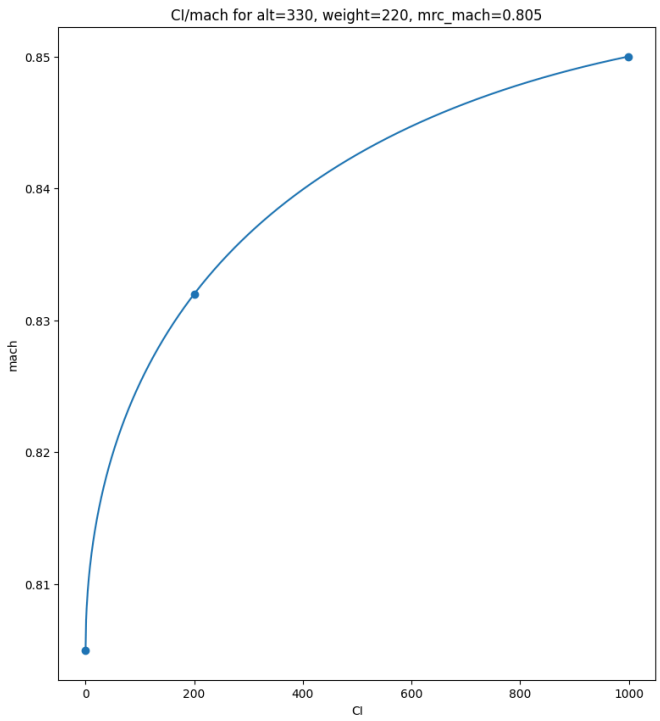




Sanity check: 0.807 MRC and 0.83 LRC

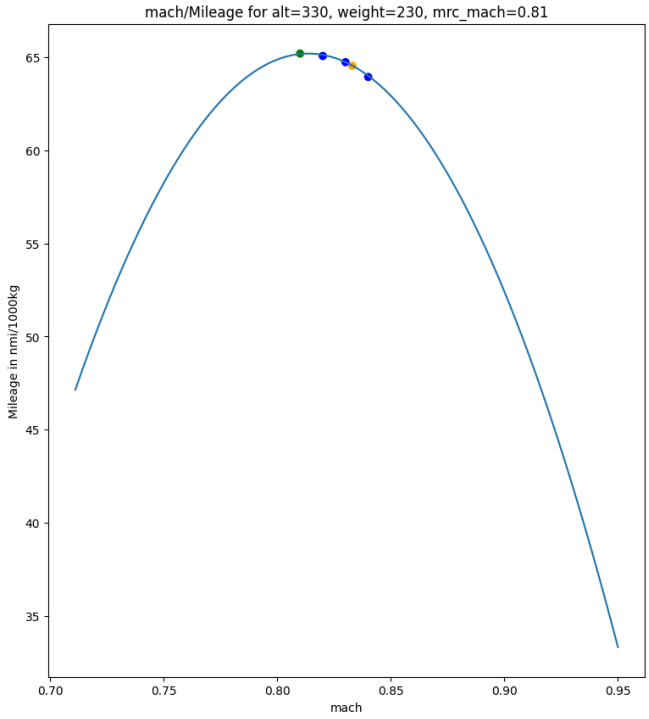
Found: 0.805 MRC and 0.832 LRC

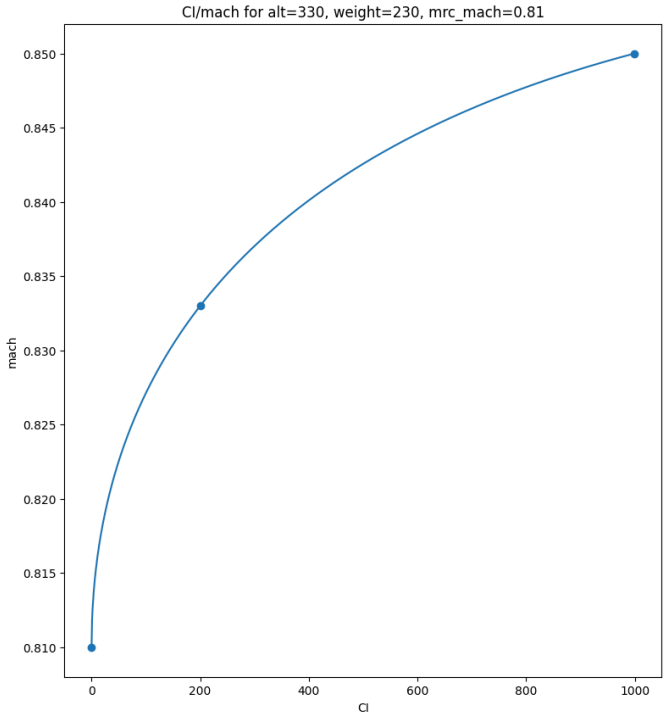




Sanity check: 0.805 MRC and 0.832 LRC

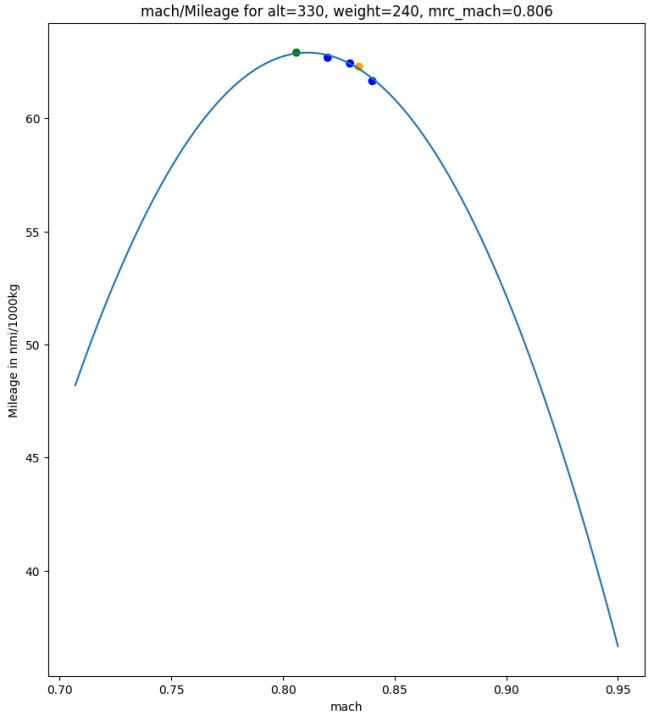
Found: 0.81 MRC and 0.833 LRC

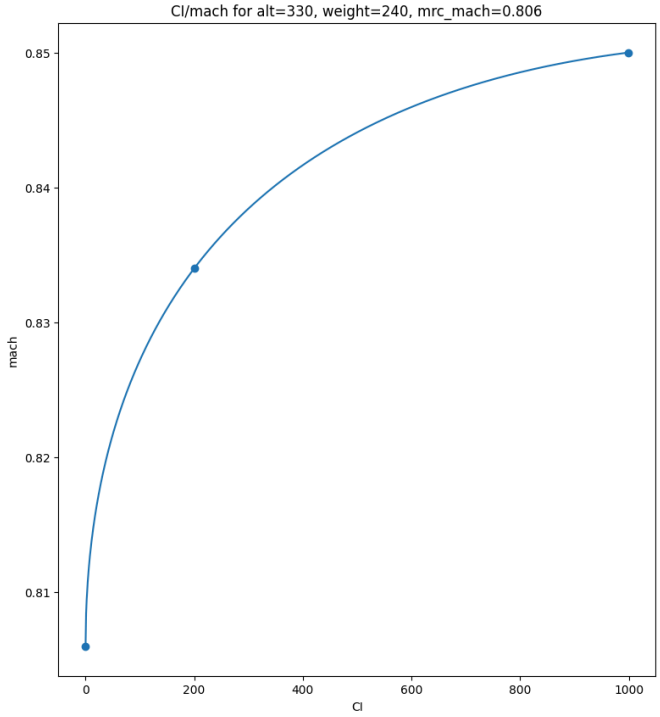




Sanity check: 0.81 MRC and 0.833 LRC

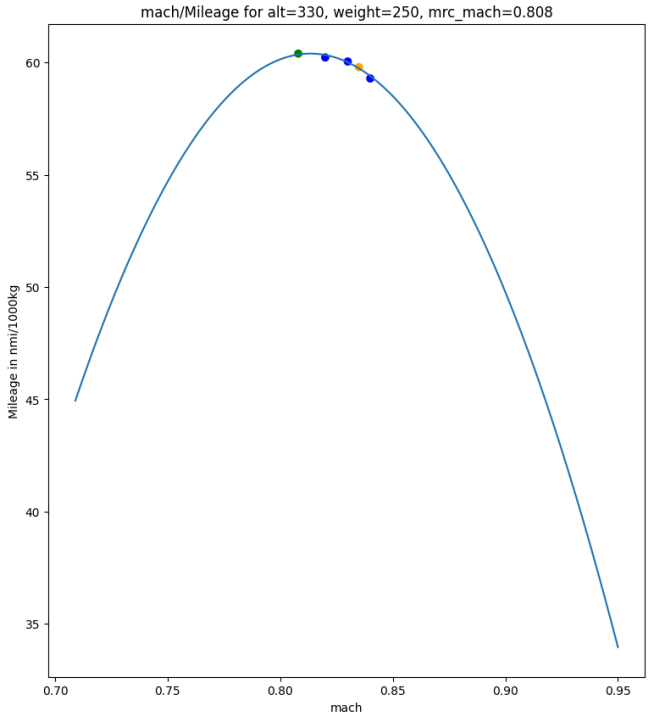
Found: 0.806 MRC and 0.834 LRC

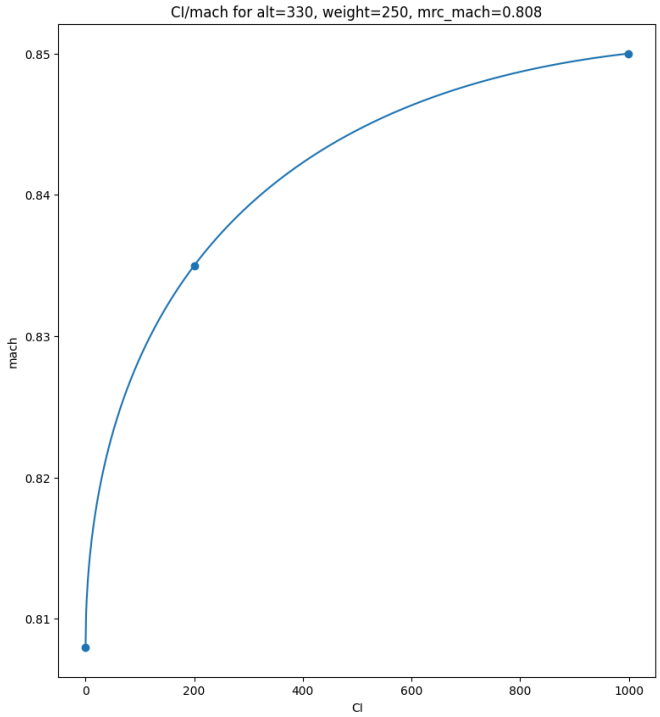




Sanity check: 0.806 MRC and 0.834 LRC

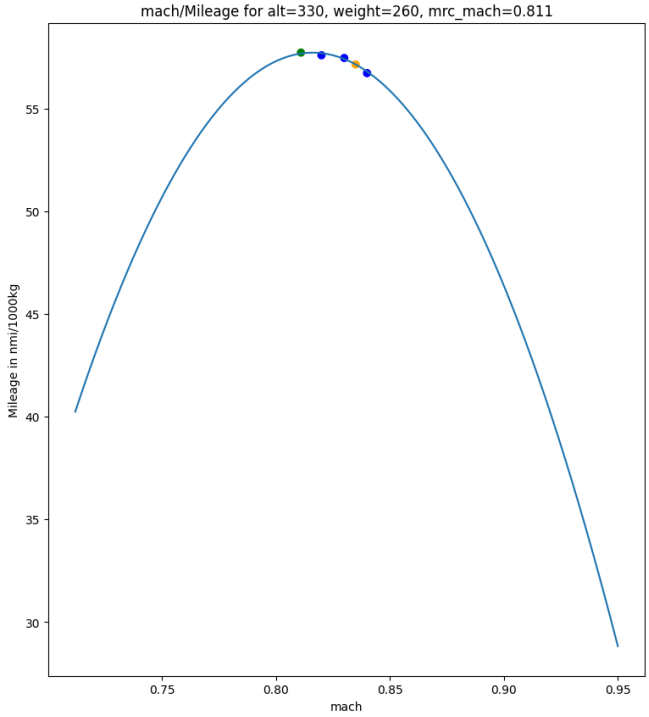
Found: 0.808 MRC and 0.835 LRC

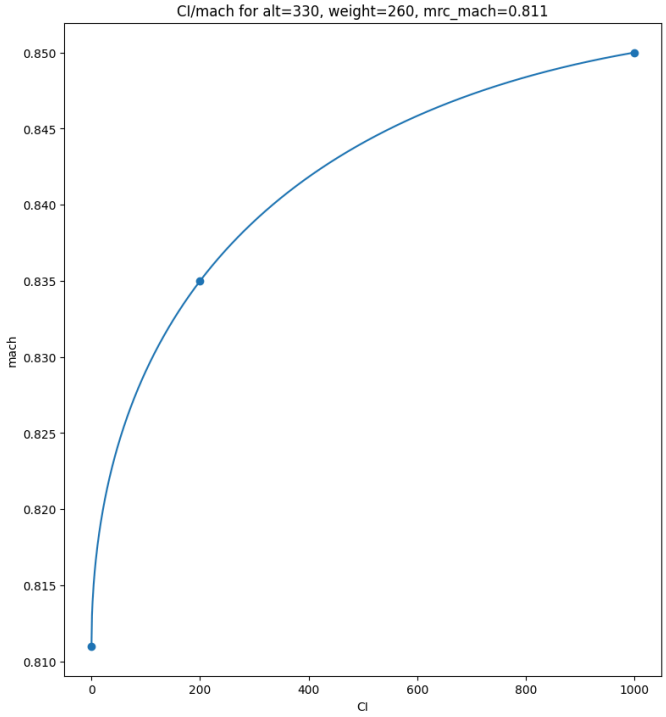




Sanity check: 0.808 MRC and 0.835 LRC

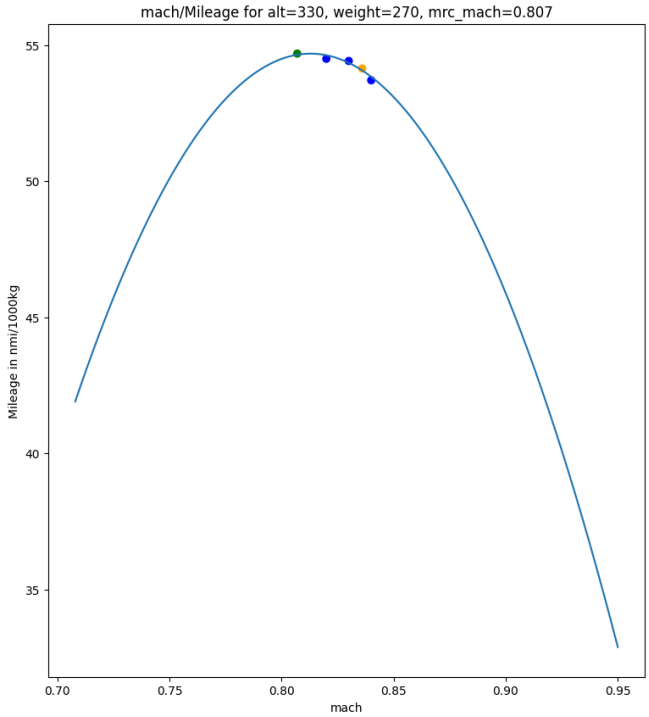
Found: 0.811 MRC and 0.835 LRC

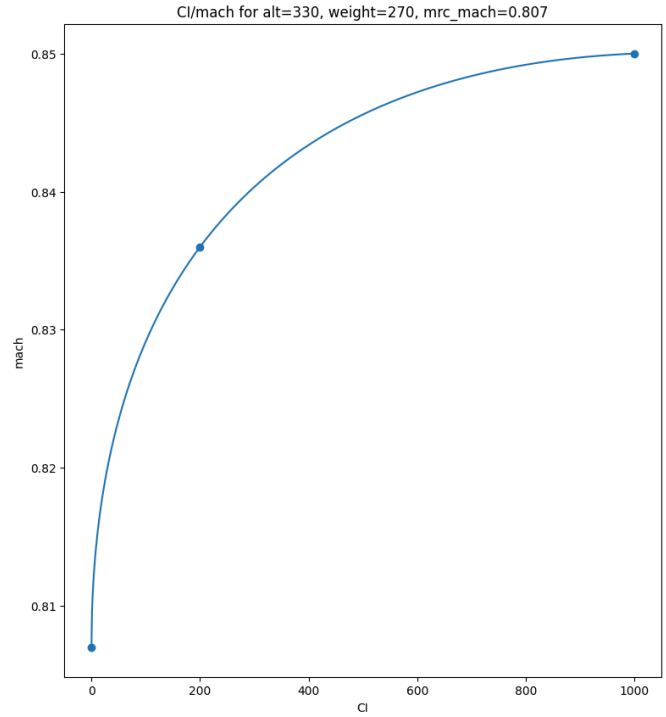




Sanity check: 0.811 MRC and 0.835 LRC

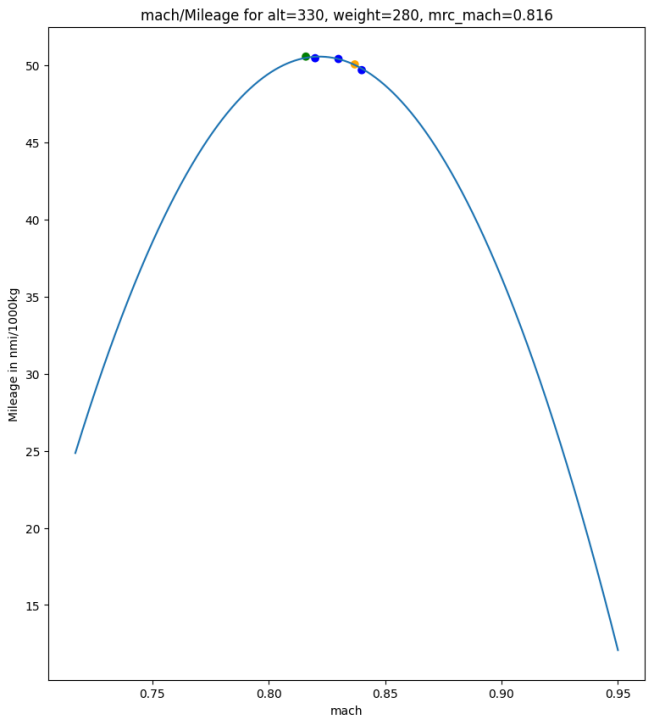
Found: 0.807 MRC and 0.836 LRC

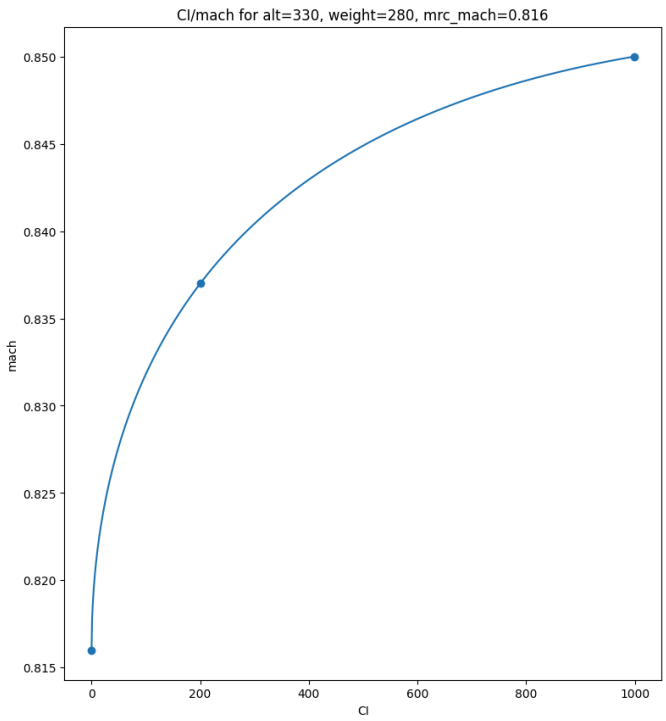




Sanity check: 0.807 MRC and 0.836 LRC

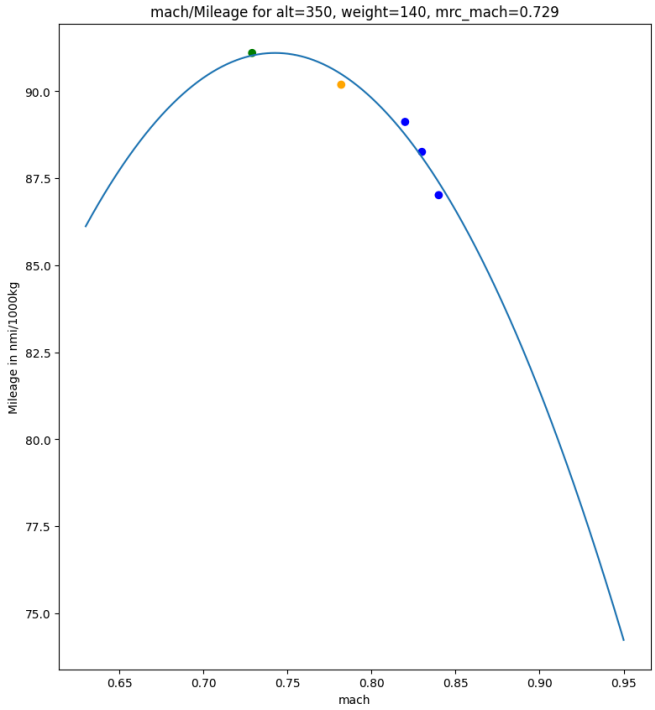
Found: 0.816 MRC and 0.837 LRC

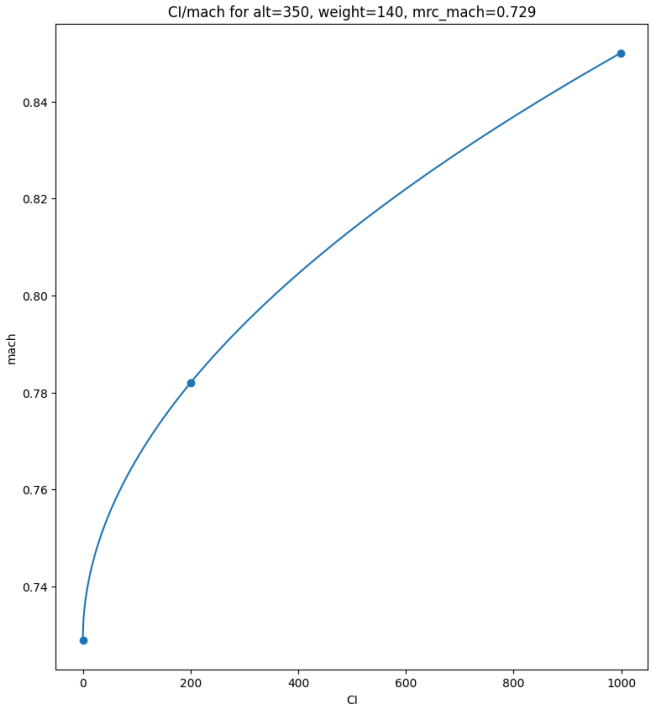




Sanity check: 0.816 MRC and 0.837 LRC

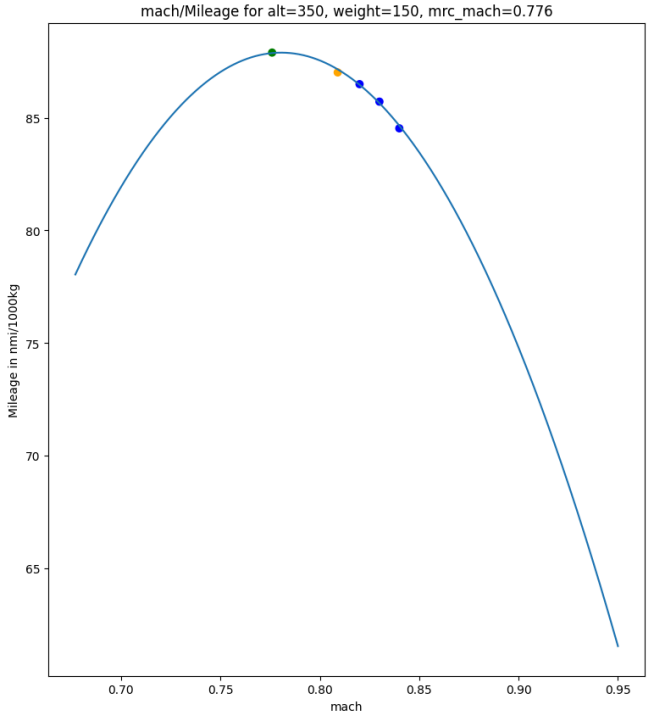
Found: 0.729 MRC and 0.782 LRC

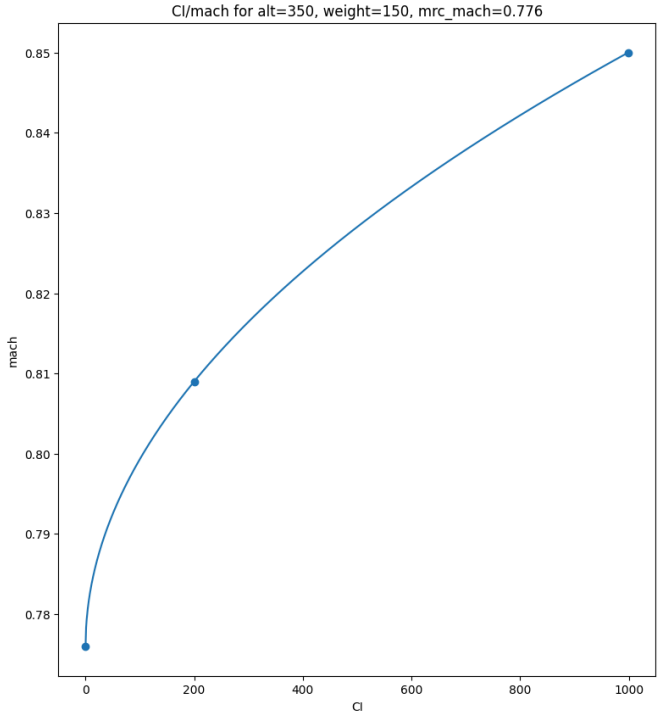




Sanity check: 0.729 MRC and 0.782 LRC

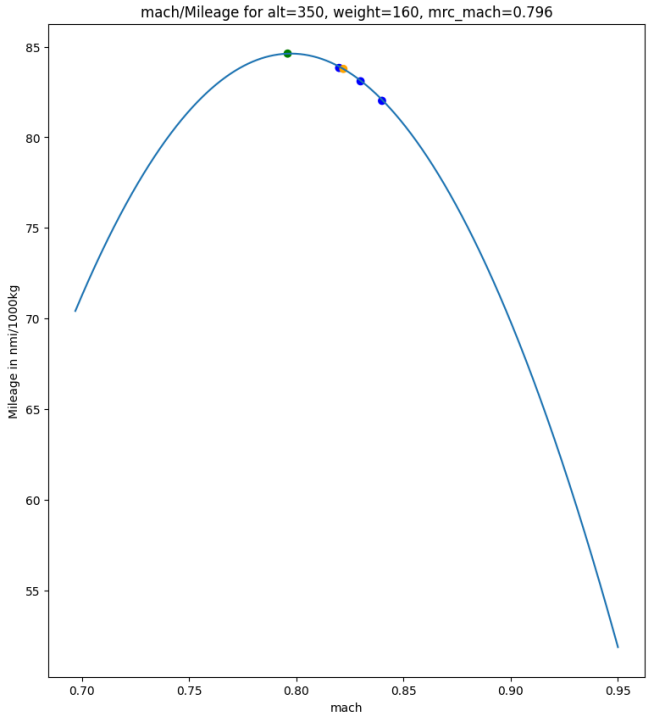
Found: 0.776 MRC and 0.809 LRC

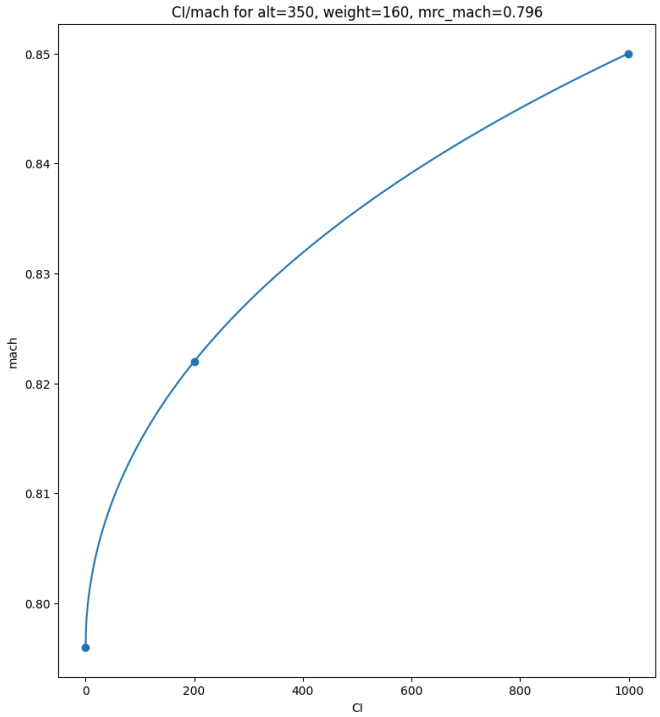




Sanity check: 0.776 MRC and 0.809 LRC

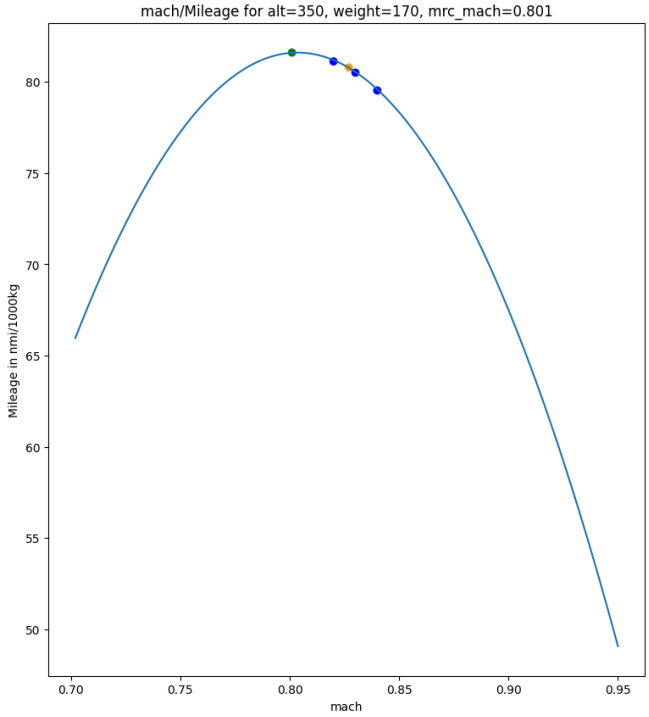
Found: 0.796 MRC and 0.822 LRC

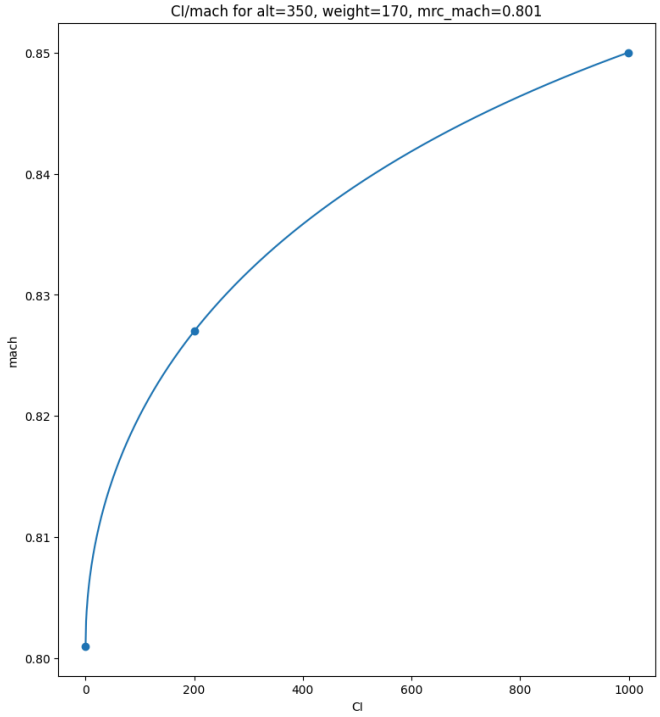




Sanity check: 0.796 MRC and 0.822 LRC

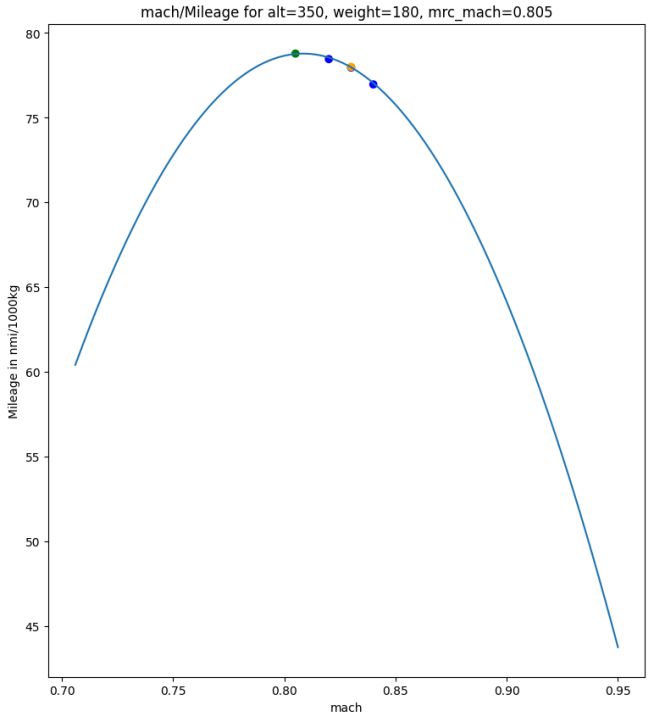
Found: 0.801 MRC and 0.827 LRC

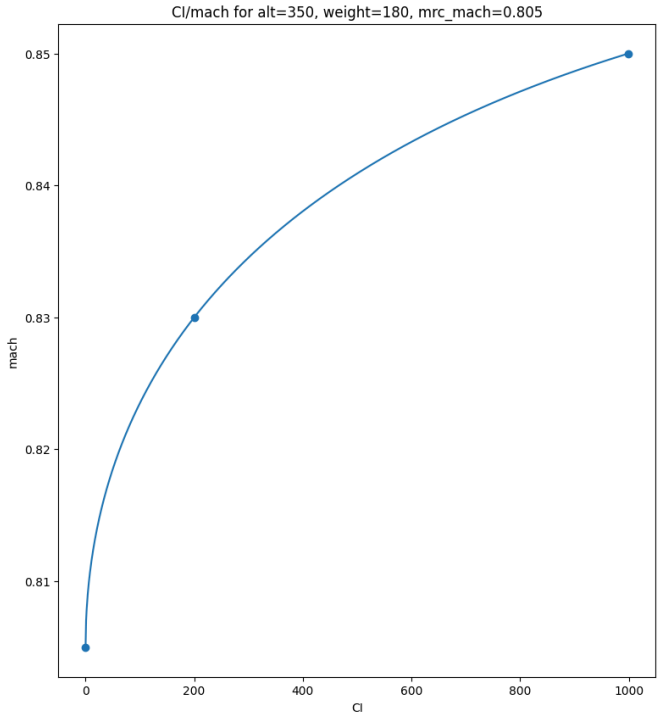




Sanity check: 0.801 MRC and 0.827 LRC

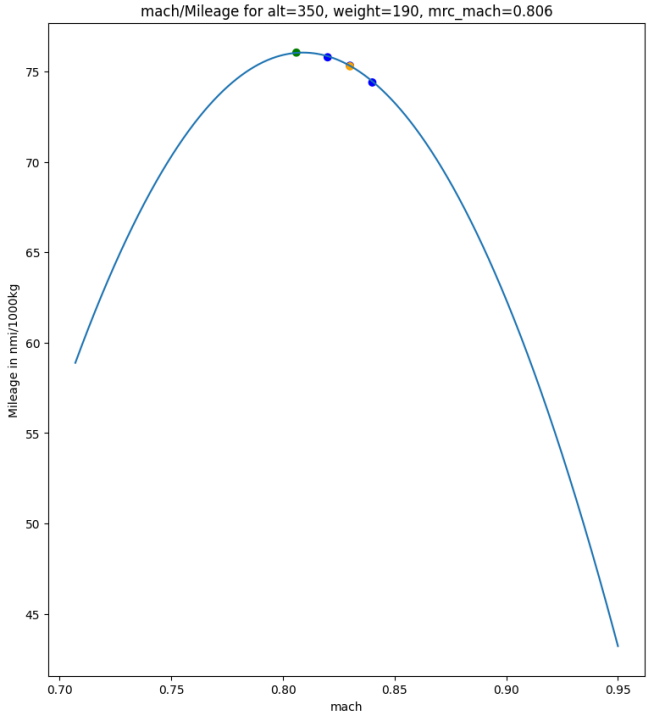
Found: 0.805 MRC and 0.83 LRC

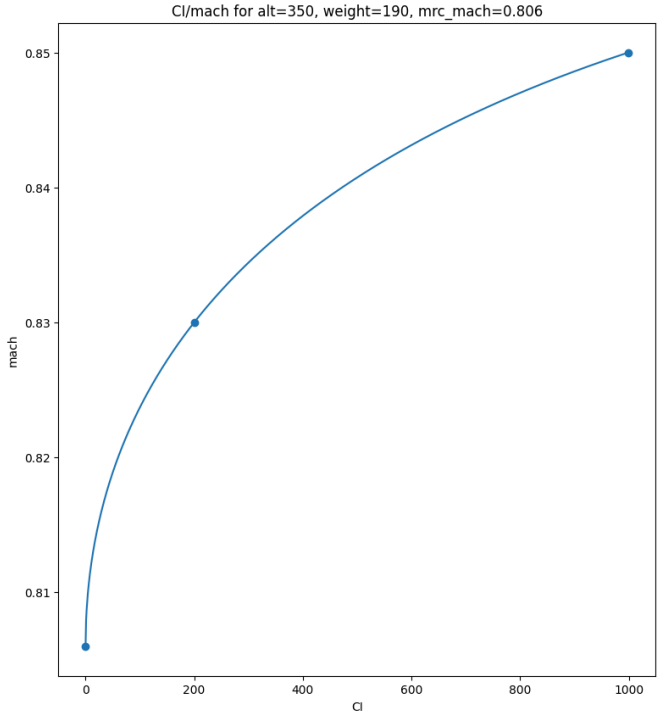




Sanity check: 0.805 MRC and 0.83 LRC

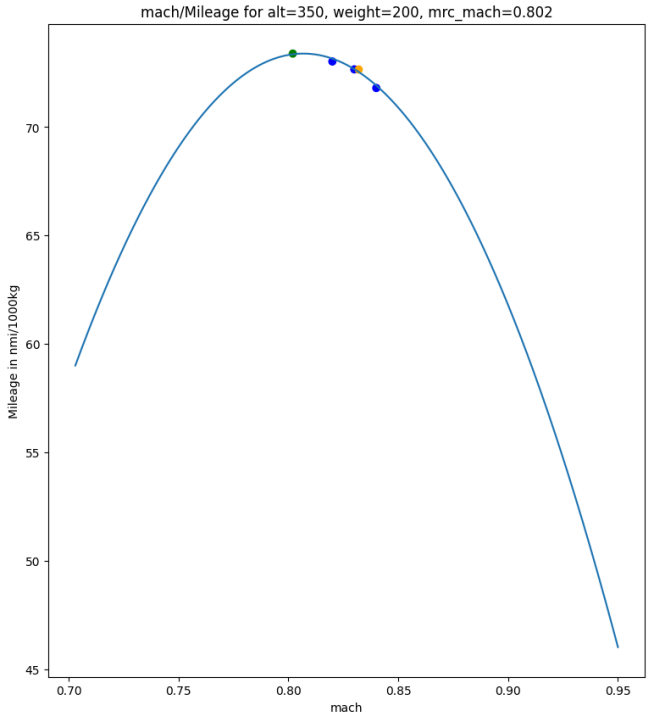
Found: 0.806 MRC and 0.83 LRC

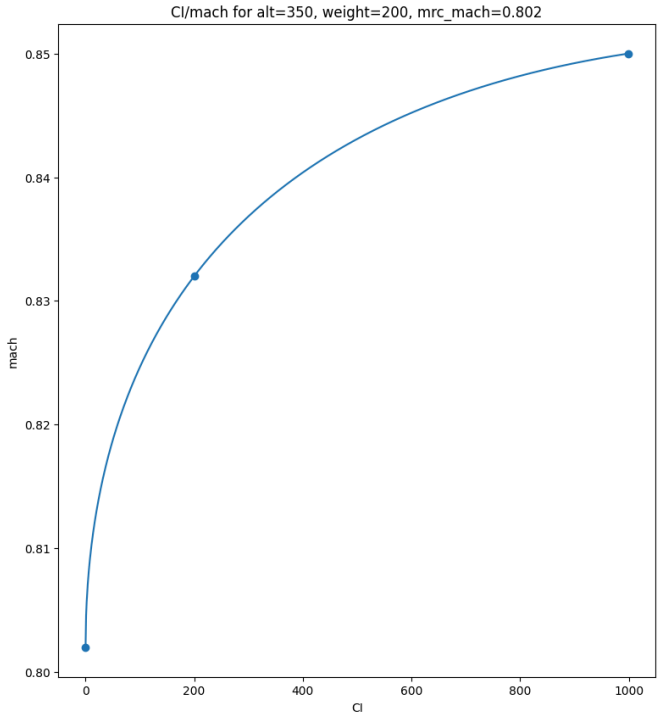




Sanity check: 0.806 MRC and 0.83 LRC

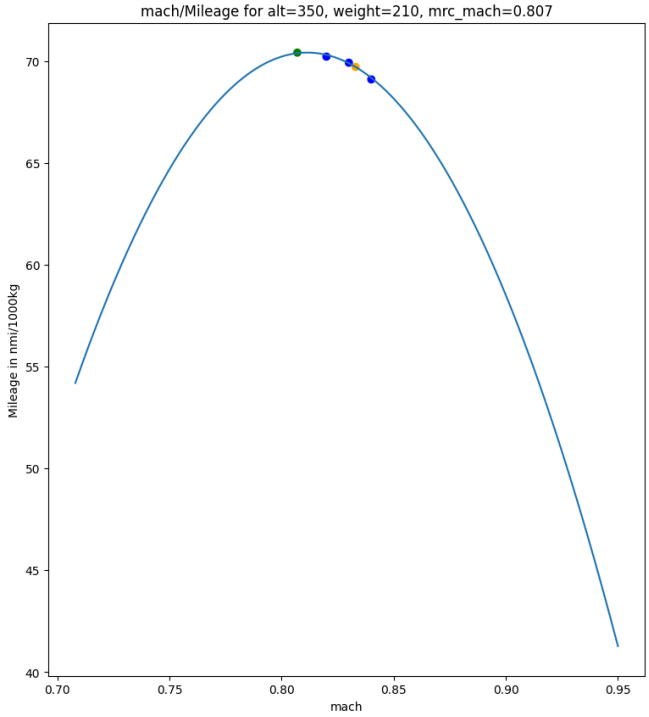
Found: 0.802 MRC and 0.832 LRC

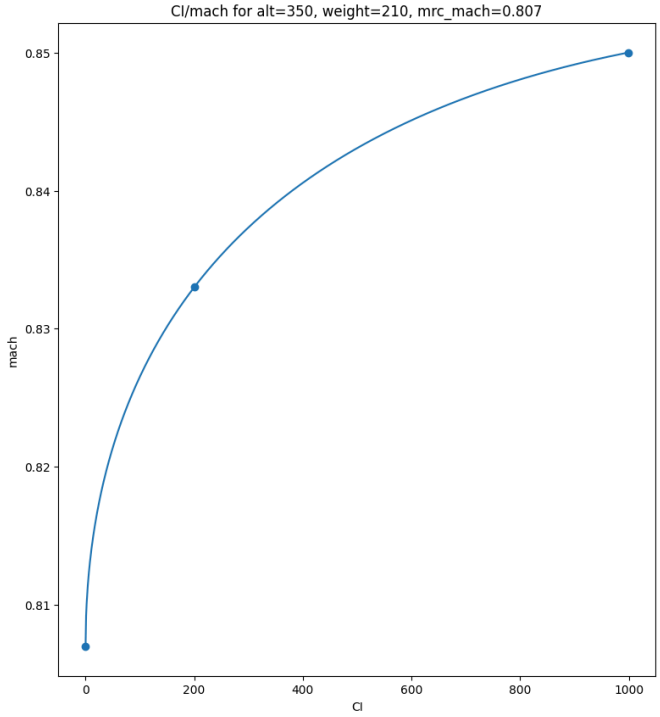




Sanity check: 0.802 MRC and 0.832 LRC

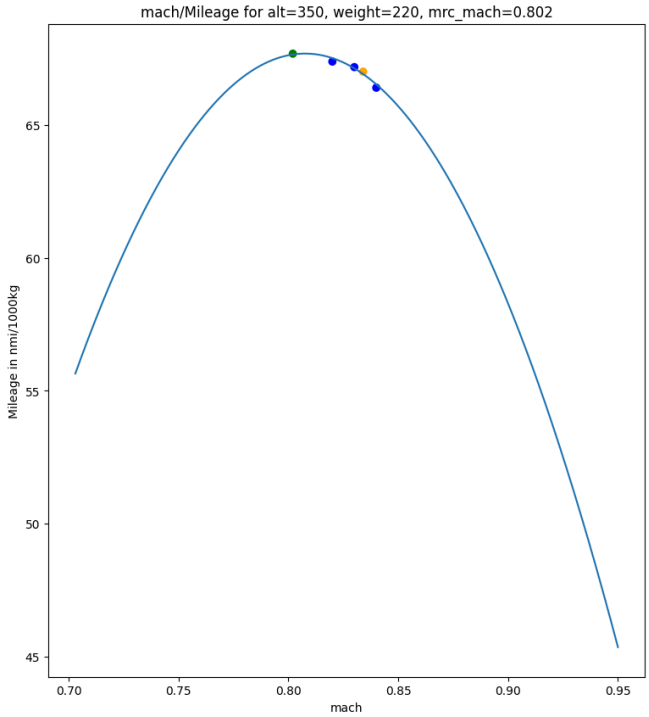
Found: 0.807 MRC and 0.833 LRC

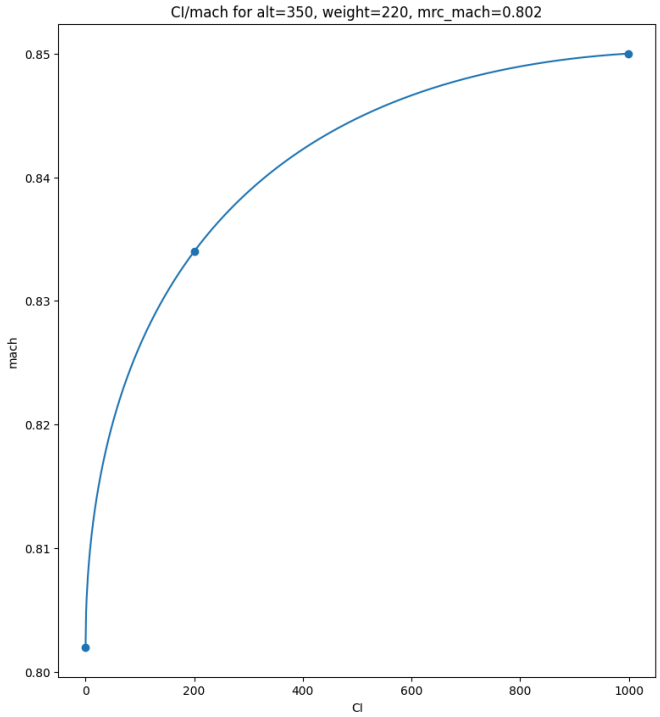




Sanity check: 0.807 MRC and 0.833 LRC

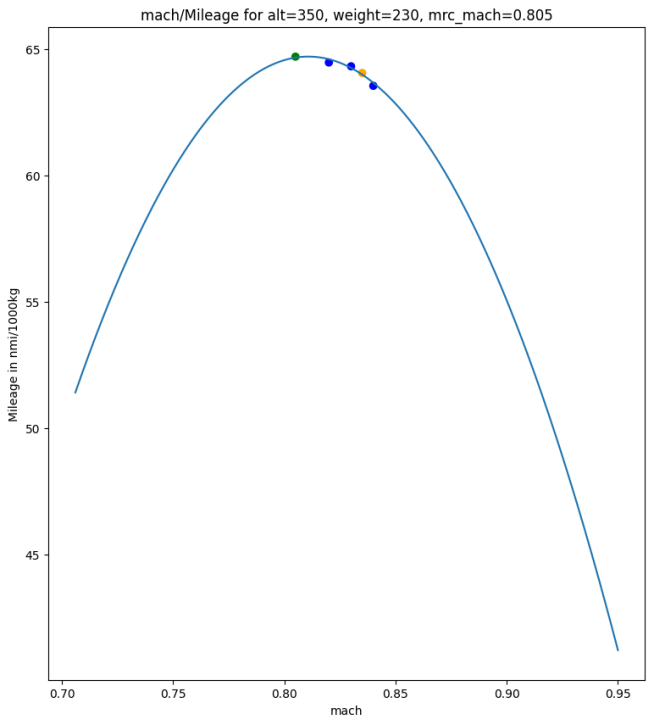
Found: 0.802 MRC and 0.834 LRC

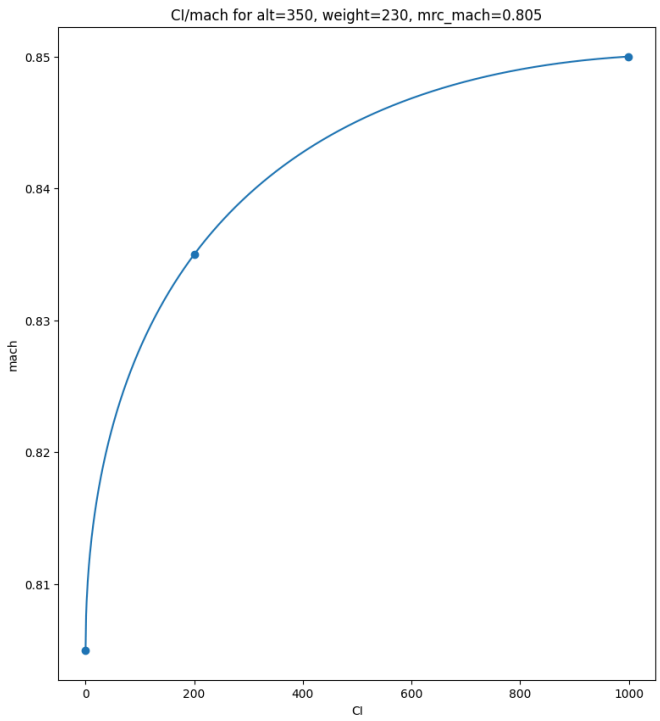




Sanity check: 0.802 MRC and 0.834 LRC

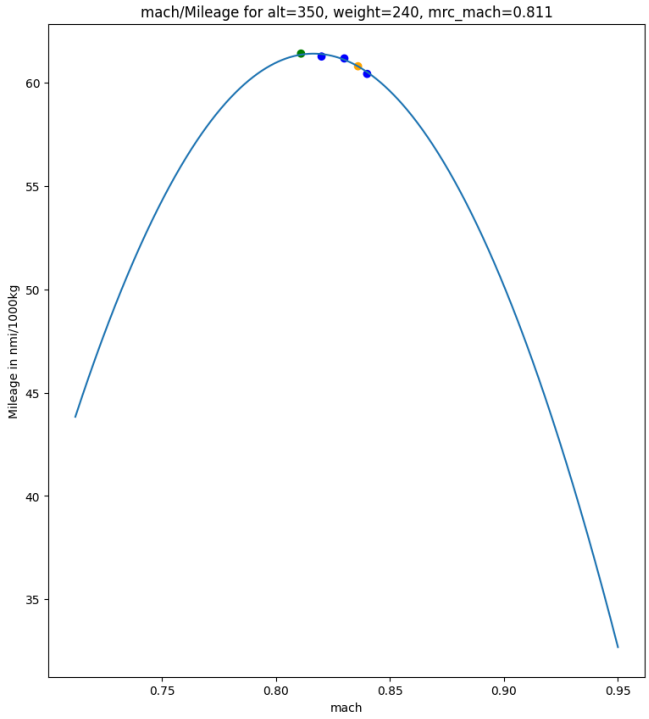
Found: 0.805 MRC and 0.835 LRC

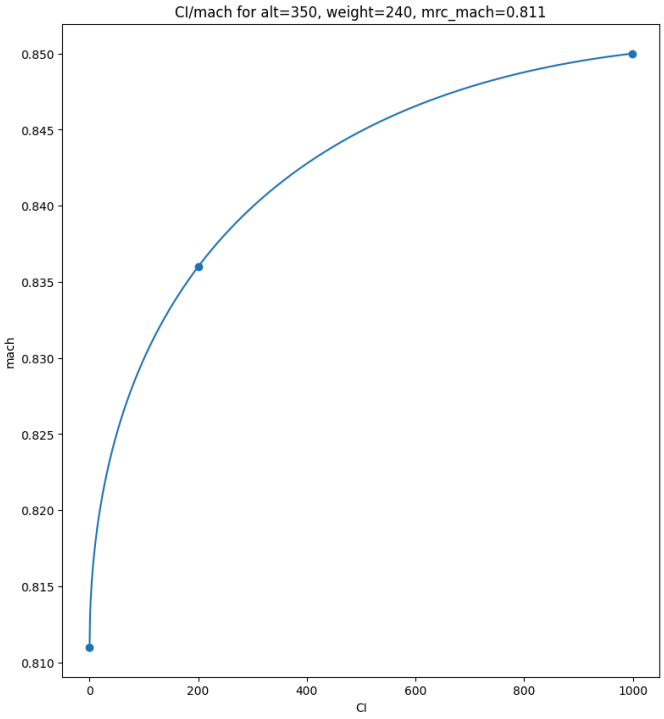




Sanity check: 0.805 MRC and 0.835 LRC

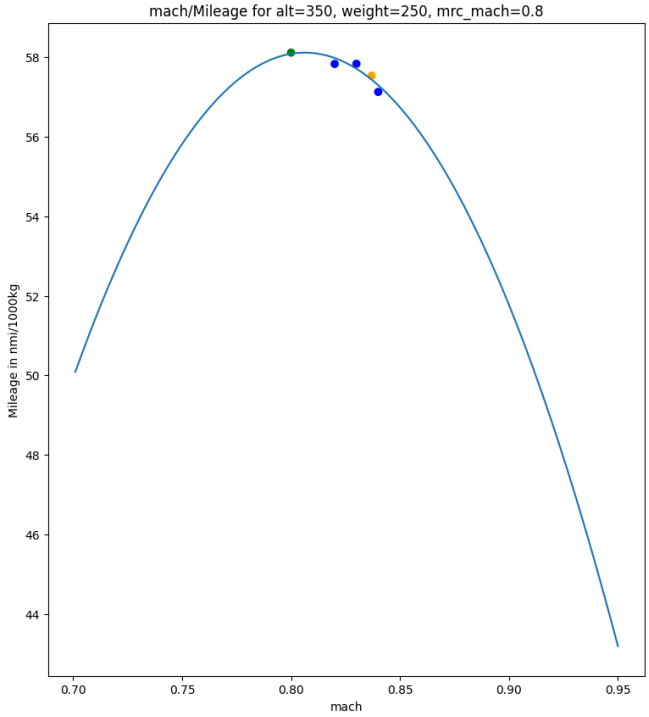
Found: 0.811 MRC and 0.836 LRC

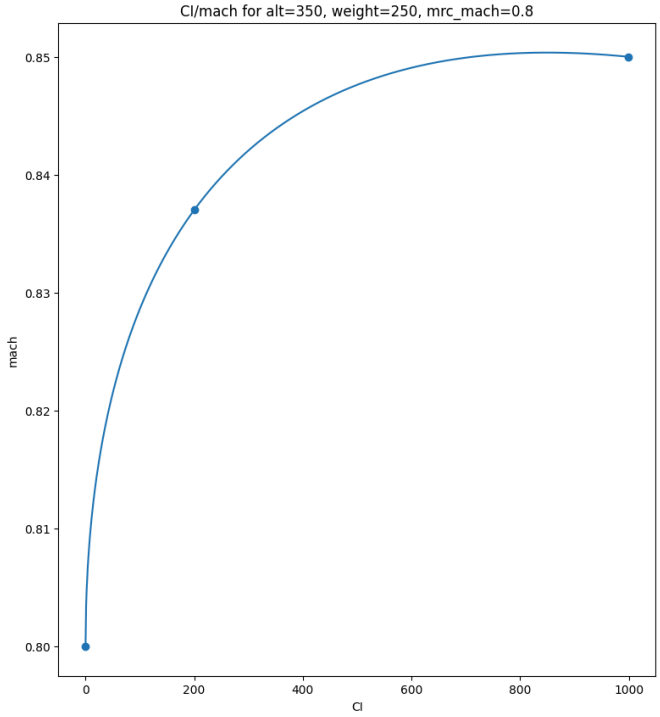




Sanity check: 0.811 MRC and 0.836 LRC

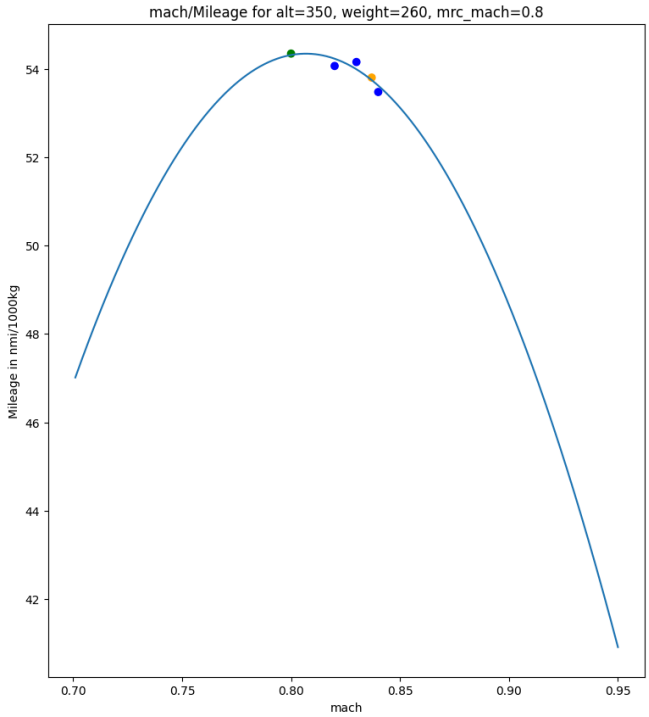
Found: 0.8 MRC and 0.837 LRC

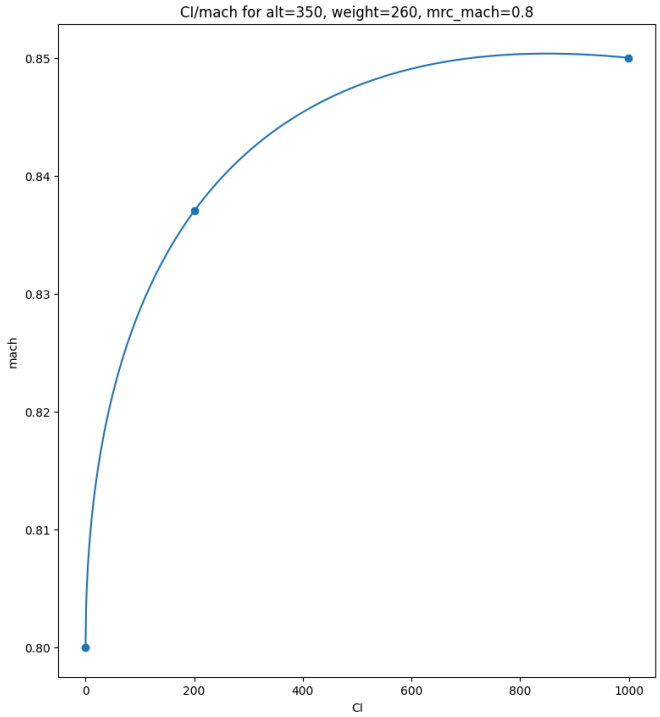




Sanity check: 0.8 MRC and 0.837 LRC

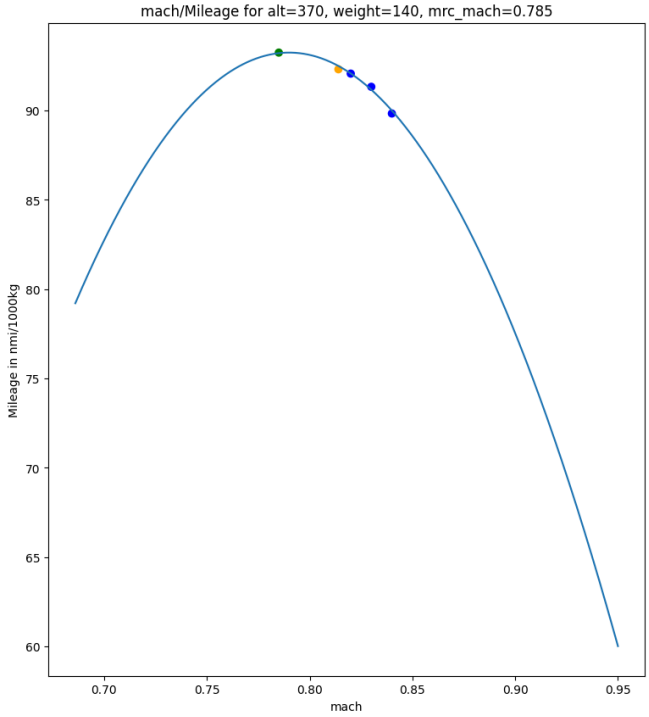
Found: 0.8 MRC and 0.837 LRC

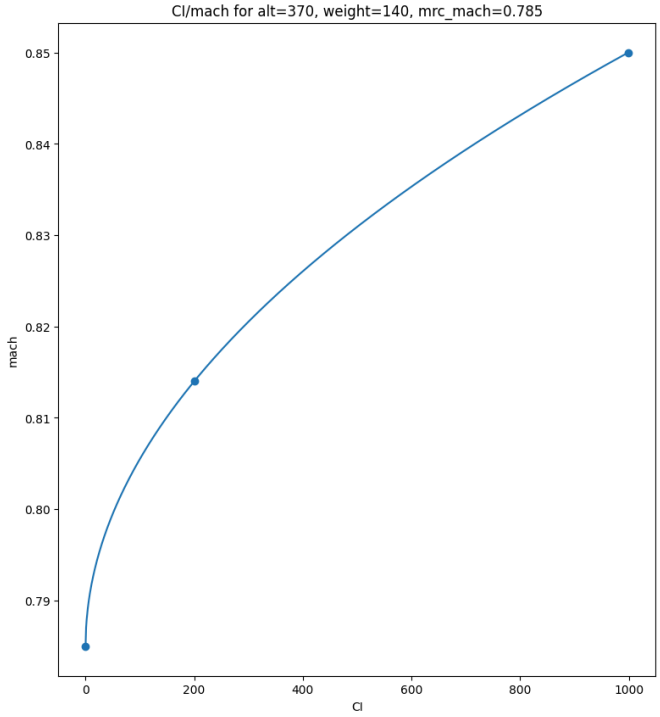




Sanity check: 0.8 MRC and 0.837 LRC

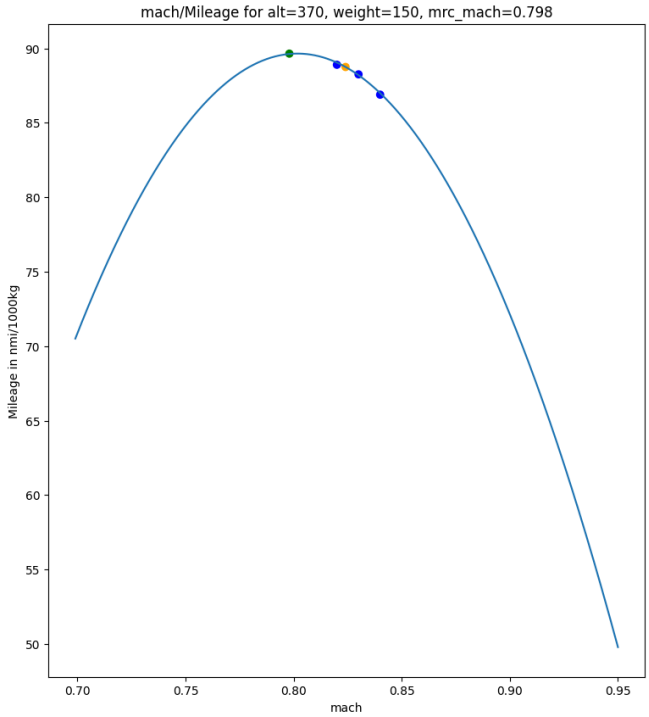
Found: 0.785 MRC and 0.814 LRC

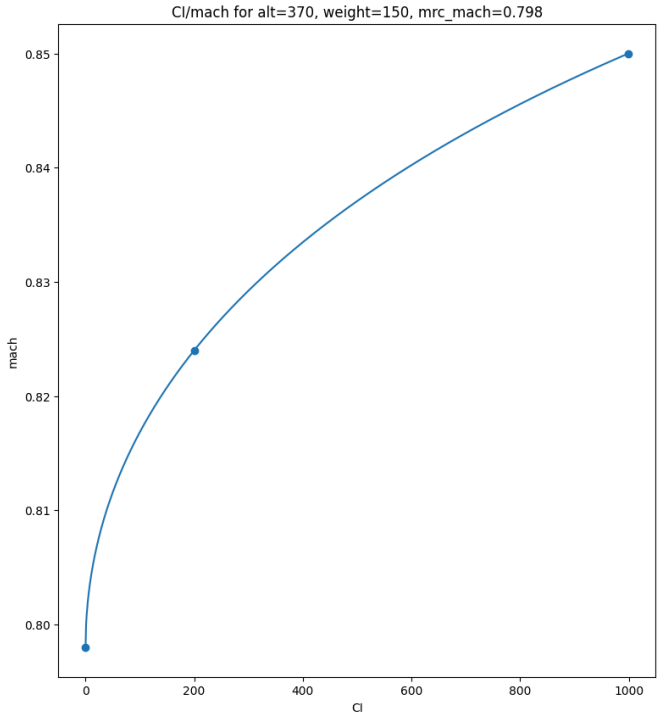




Sanity check: 0.785 MRC and 0.814 LRC

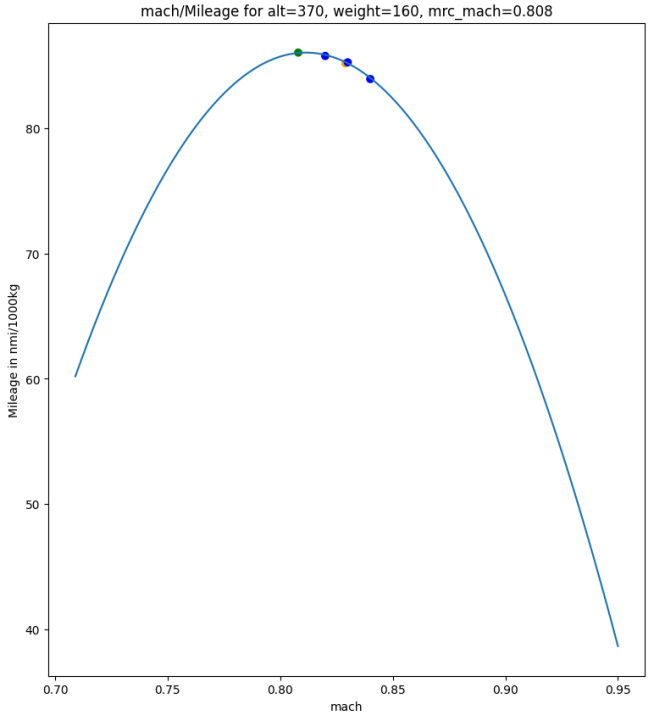
Found: 0.798 MRC and 0.824 LRC

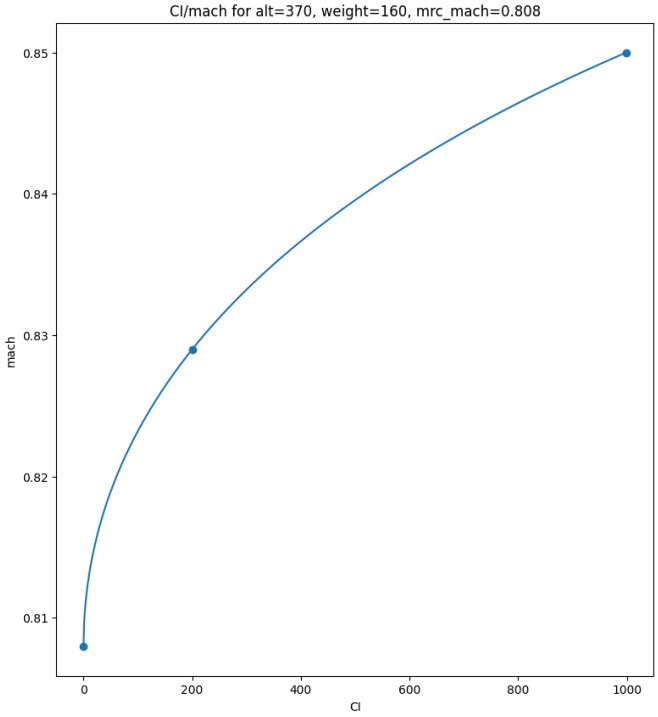




Sanity check: 0.798 MRC and 0.824 LRC

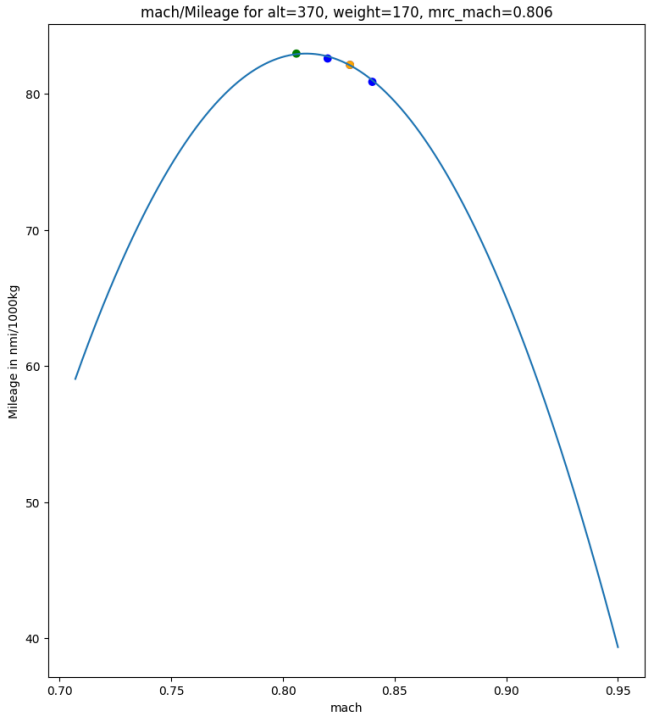
Found: 0.808 MRC and 0.829 LRC

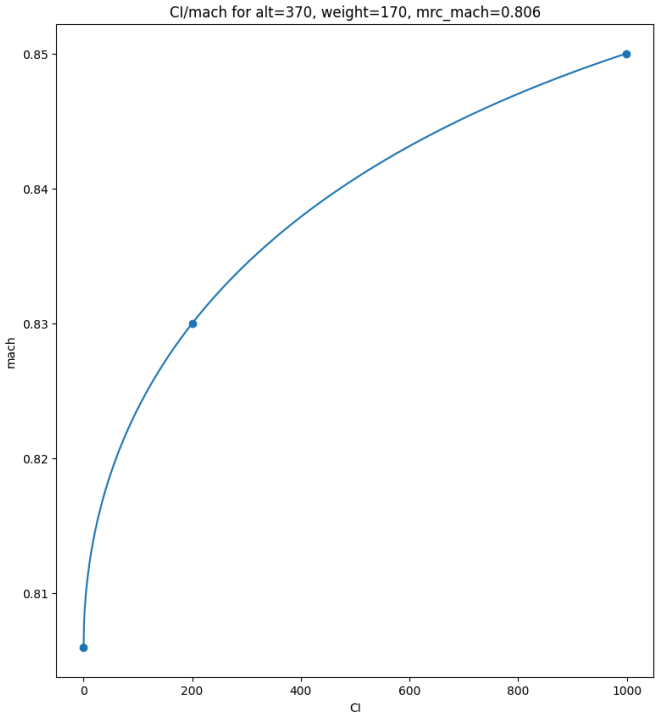




Sanity check: 0.808 MRC and 0.829 LRC

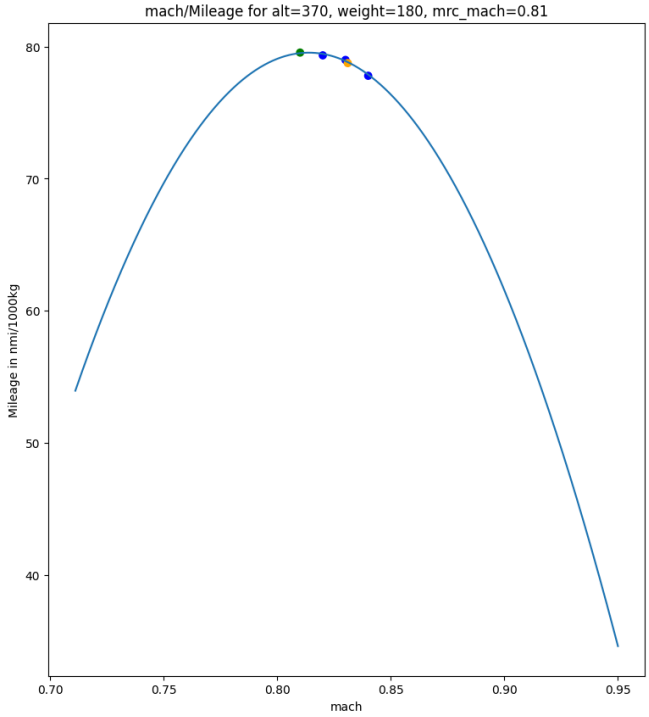
Found: 0.806 MRC and 0.83 LRC

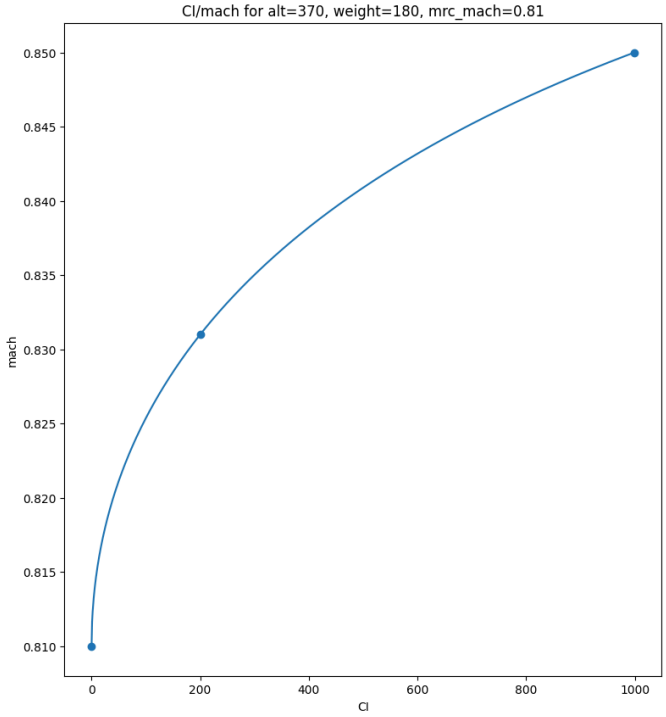




Sanity check: 0.806 MRC and 0.83 LRC

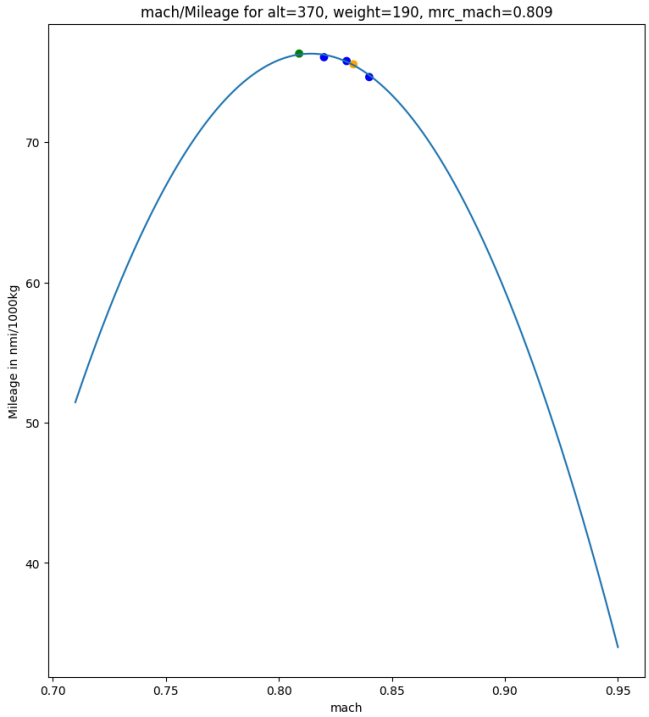
Found: 0.81 MRC and 0.831 LRC

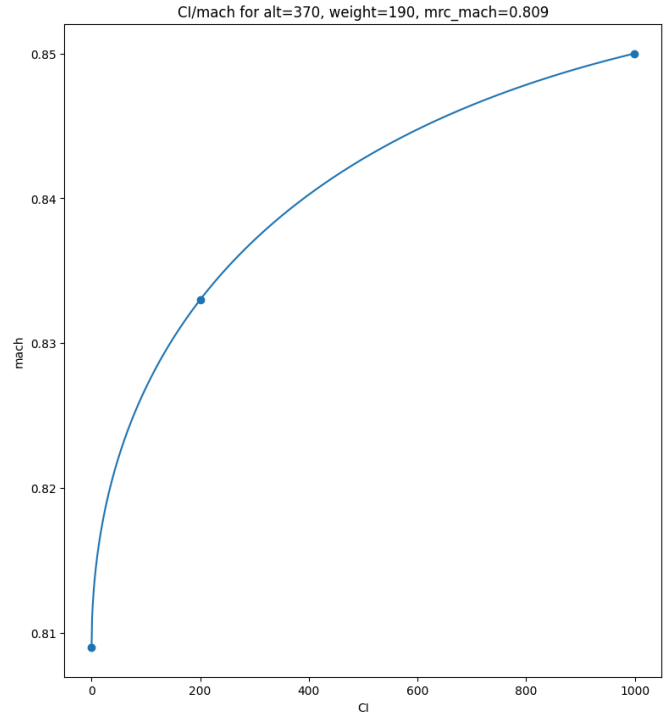




Sanity check: 0.81 MRC and 0.831 LRC

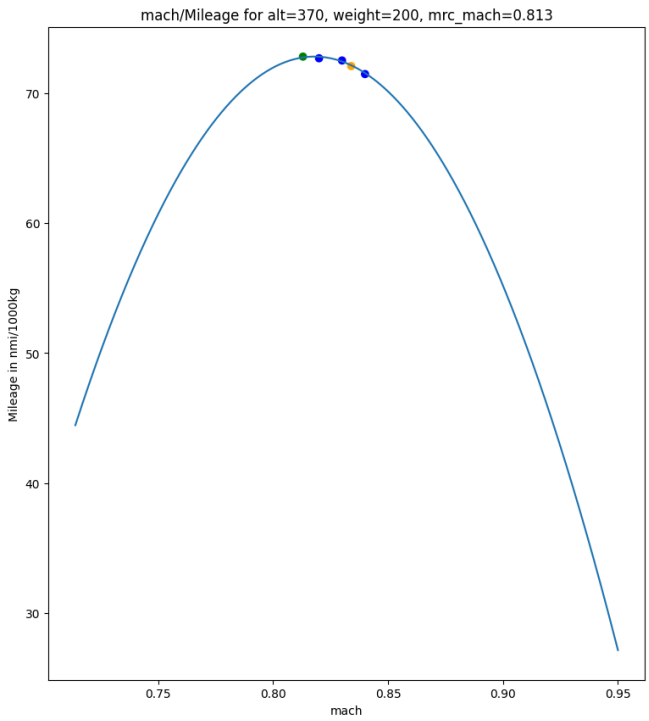
Found: 0.809 MRC and 0.833 LRC

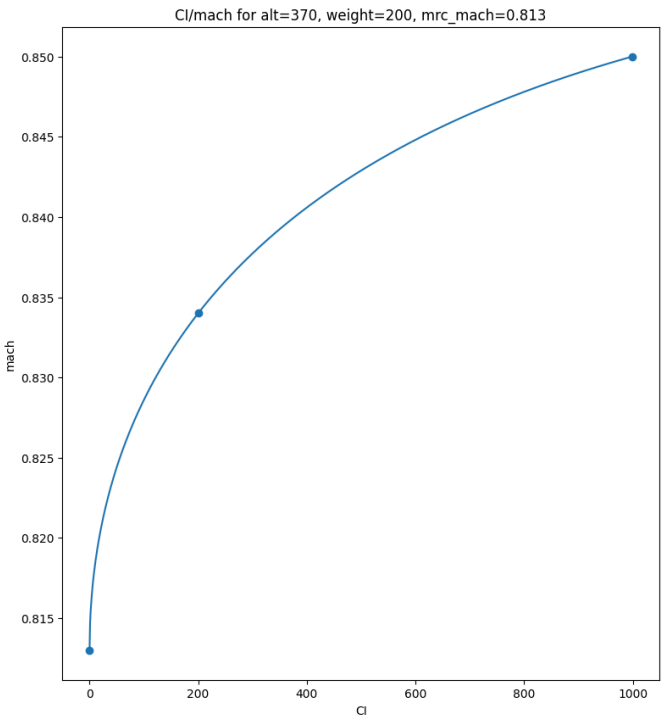




Sanity check: 0.809 MRC and 0.833 LRC

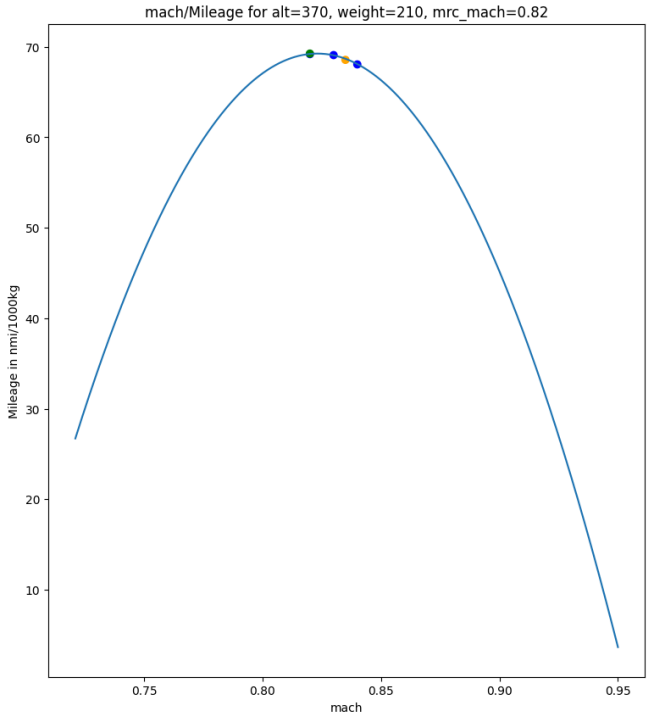
Found: 0.813 MRC and 0.834 LRC

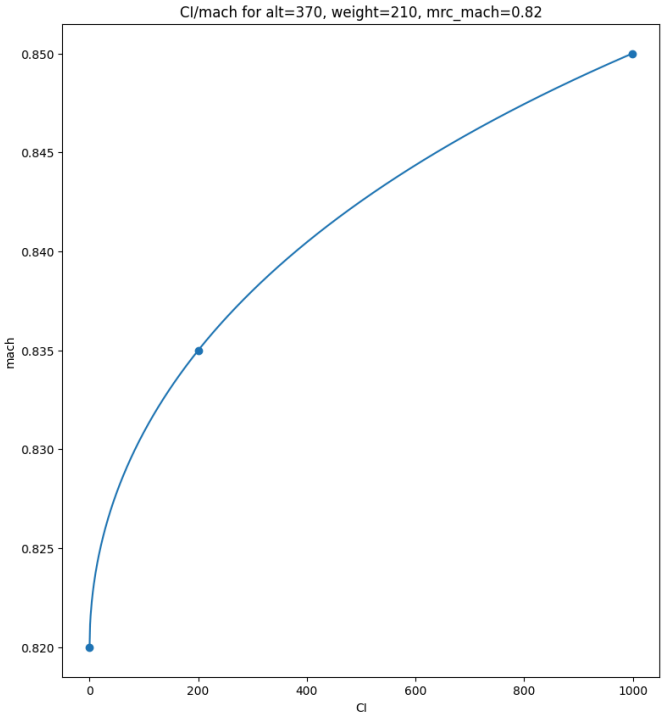




Sanity check: 0.813 MRC and 0.834 LRC

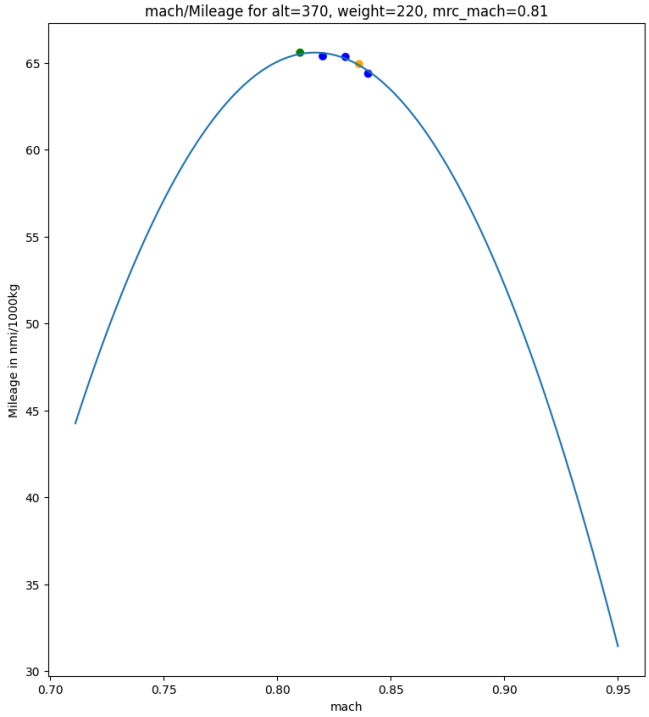
Found: 0.82 MRC and 0.835 LRC

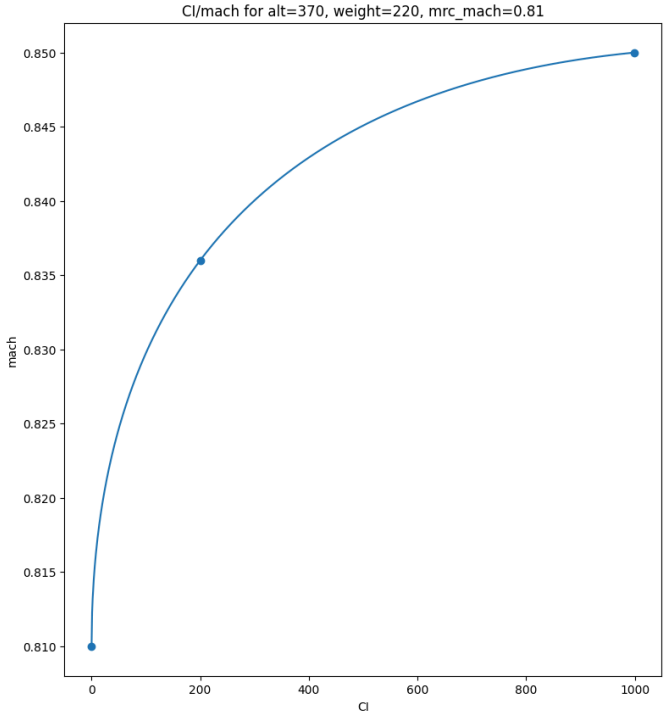




Sanity check: 0.82 MRC and 0.835 LRC

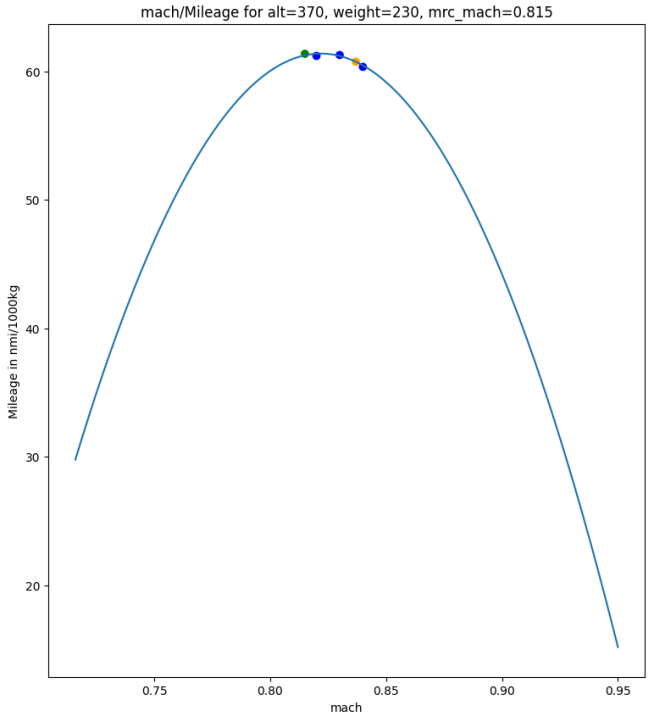
Found: 0.81 MRC and 0.836 LRC

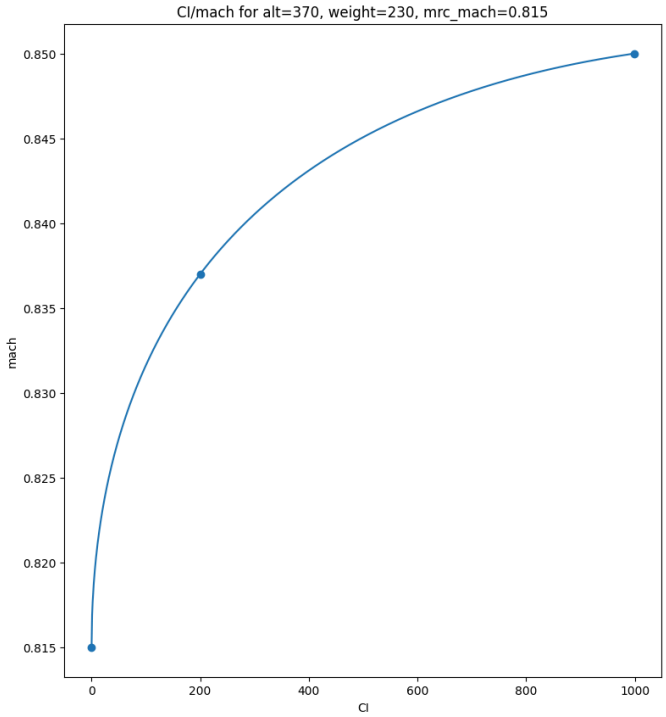




Sanity check: 0.81 MRC and 0.836 LRC

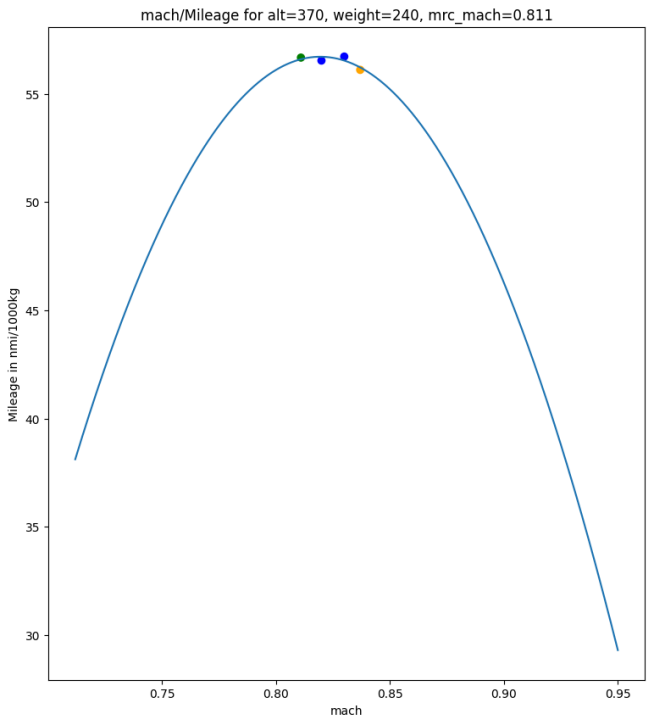
Found: 0.815 MRC and 0.837 LRC

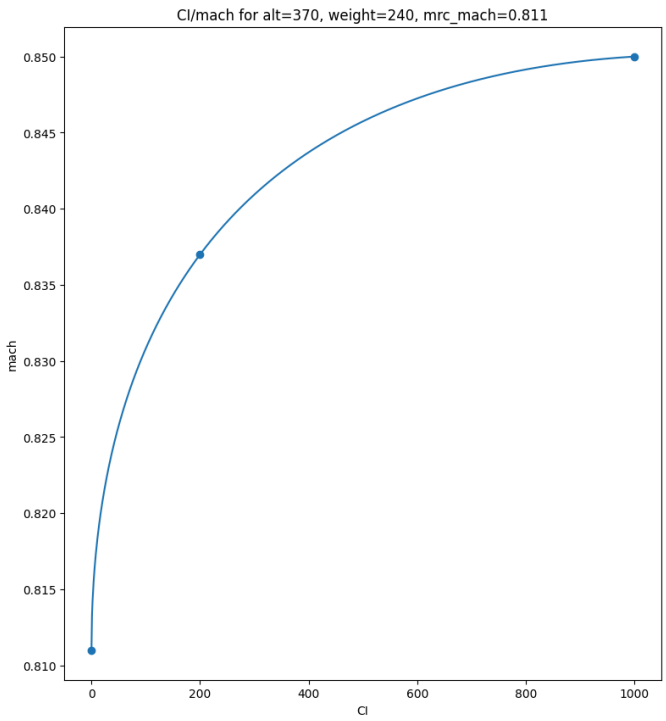




Sanity check: 0.815 MRC and 0.837 LRC

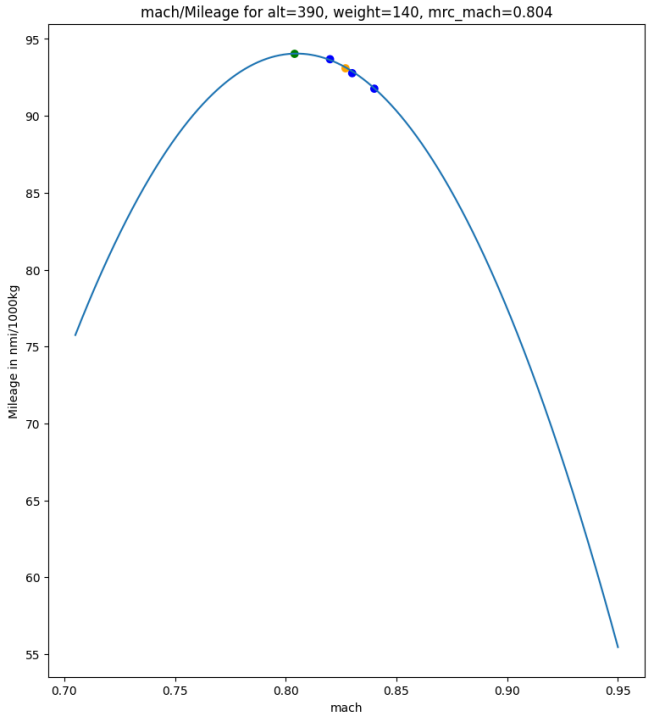
Found: 0.811 MRC and 0.837 LRC

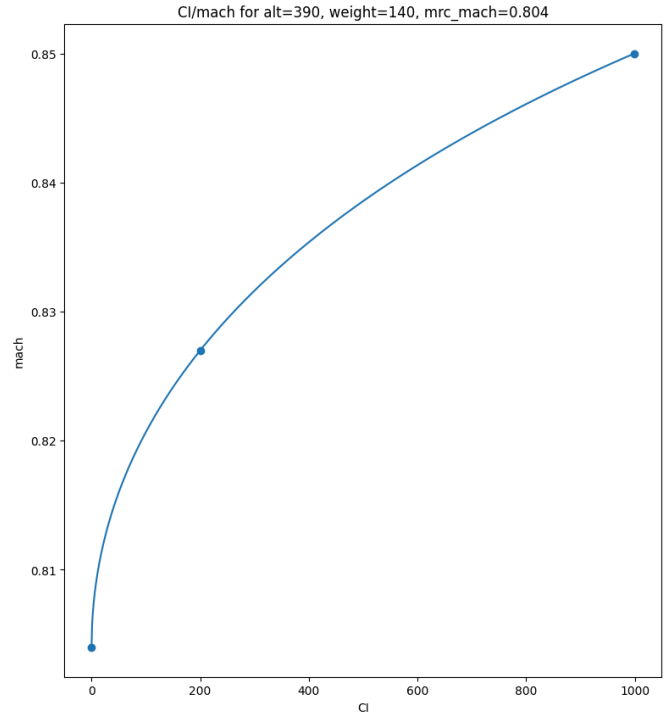




Sanity check: 0.811 MRC and 0.837 LRC

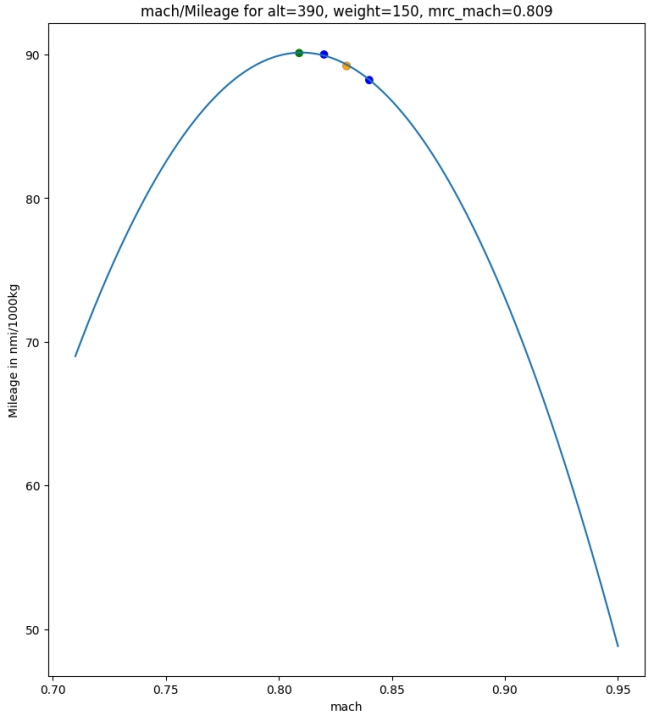
Found: 0.804 MRC and 0.827 LRC

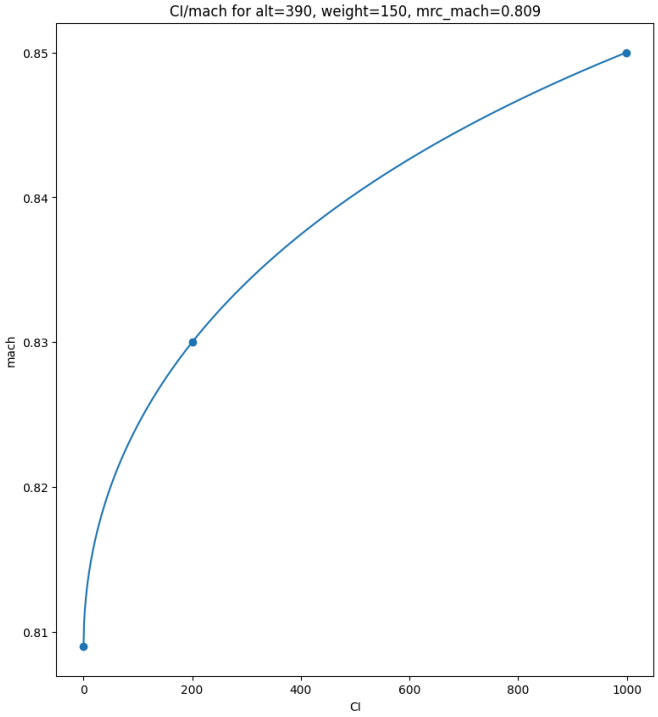




Sanity check: 0.804 MRC and 0.827 LRC

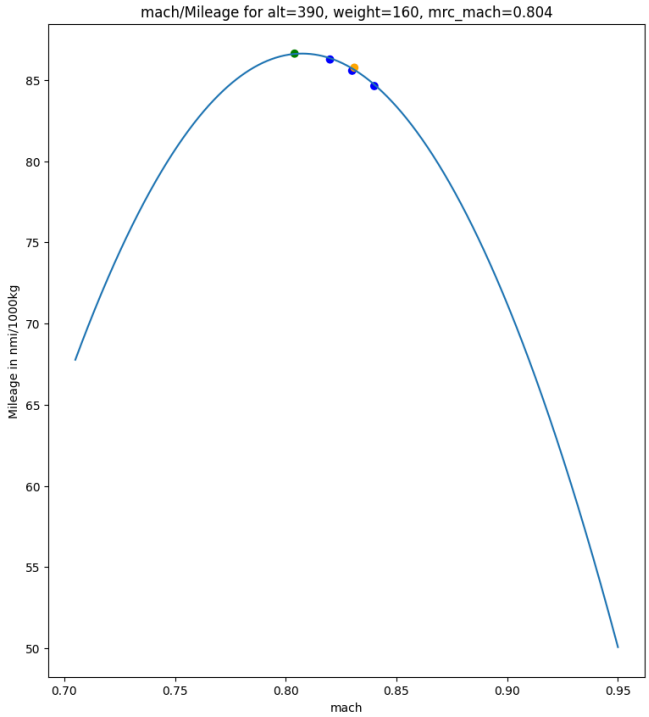
Found: 0.809 MRC and 0.83 LRC

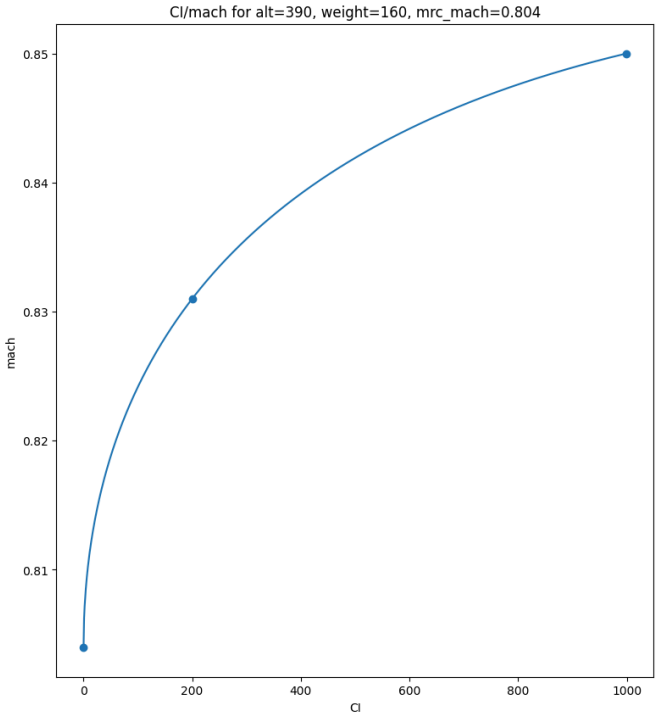




Sanity check: 0.809 MRC and 0.83 LRC

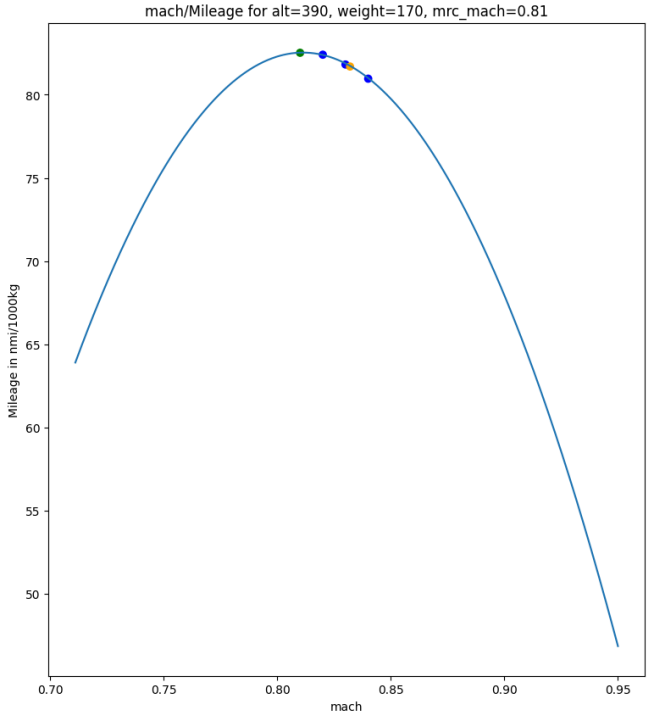
Found: 0.804 MRC and 0.831 LRC

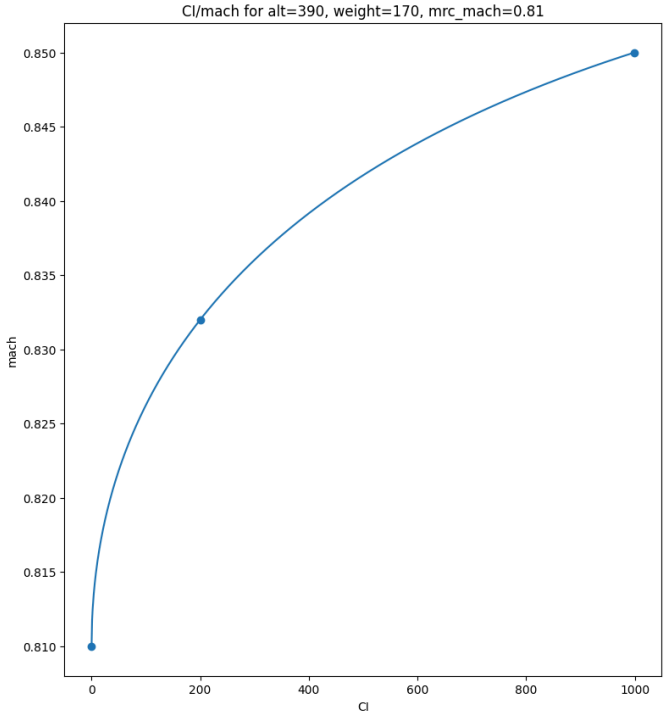




Sanity check: 0.804 MRC and 0.831 LRC

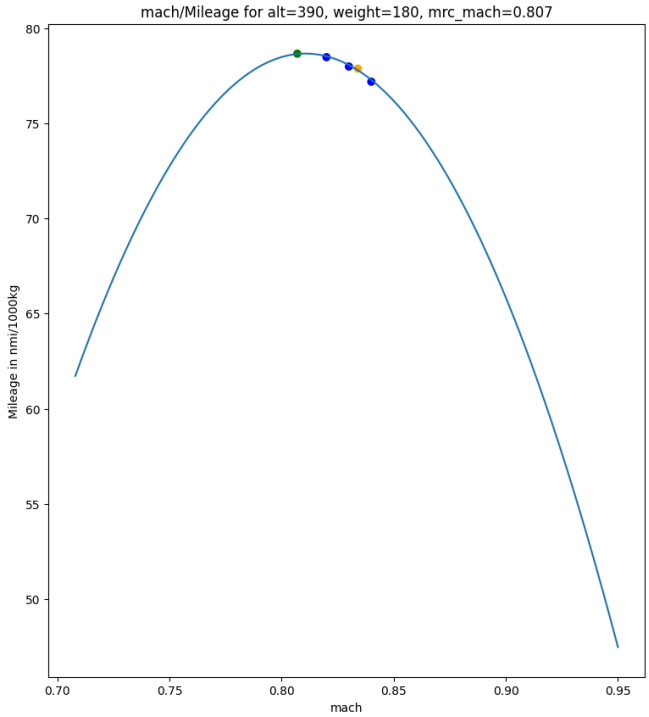
Found: 0.81 MRC and 0.832 LRC

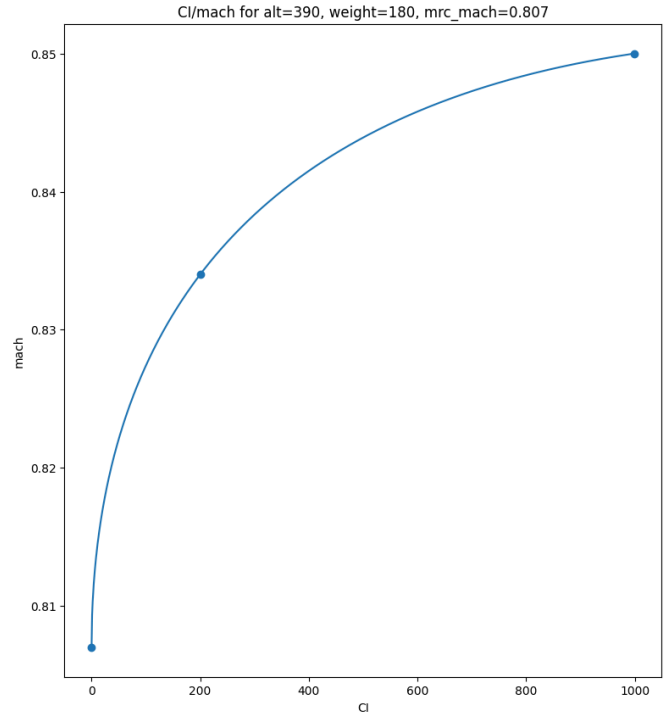




Sanity check: 0.81 MRC and 0.832 LRC

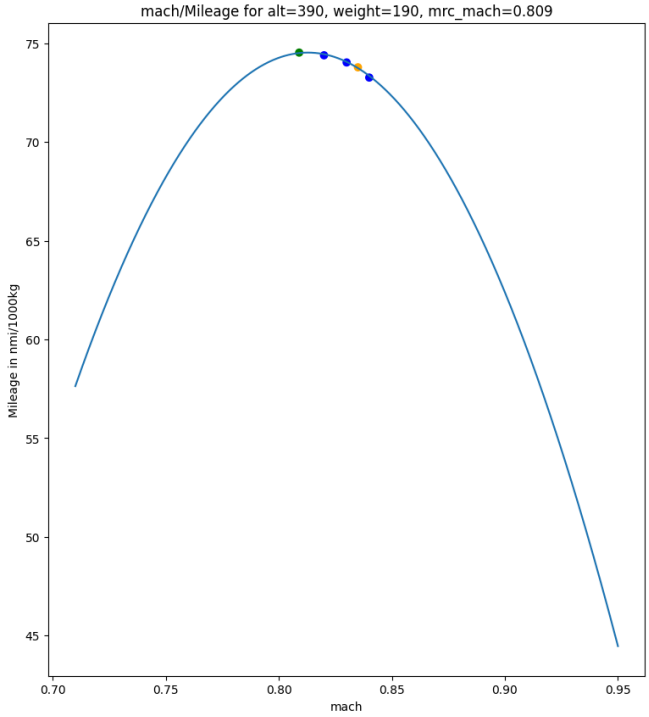
Found: 0.807 MRC and 0.834 LRC

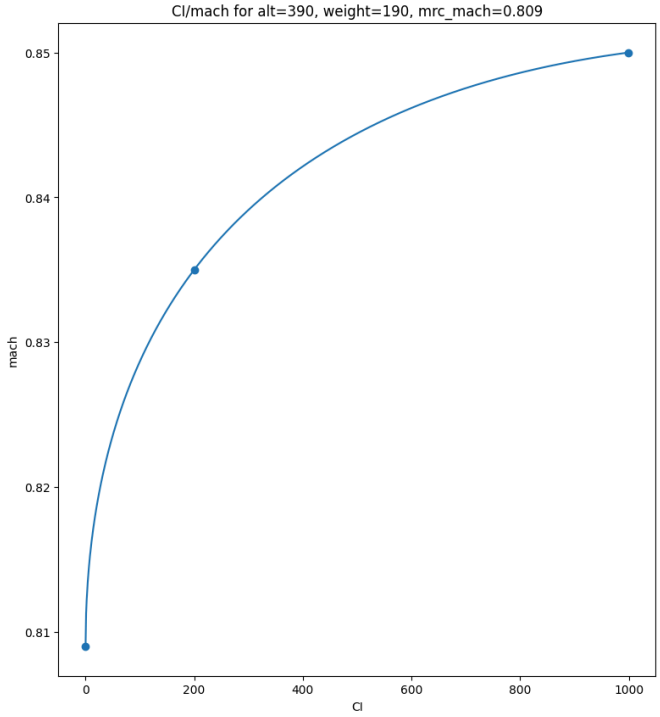




Sanity check: 0.807 MRC and 0.834 LRC

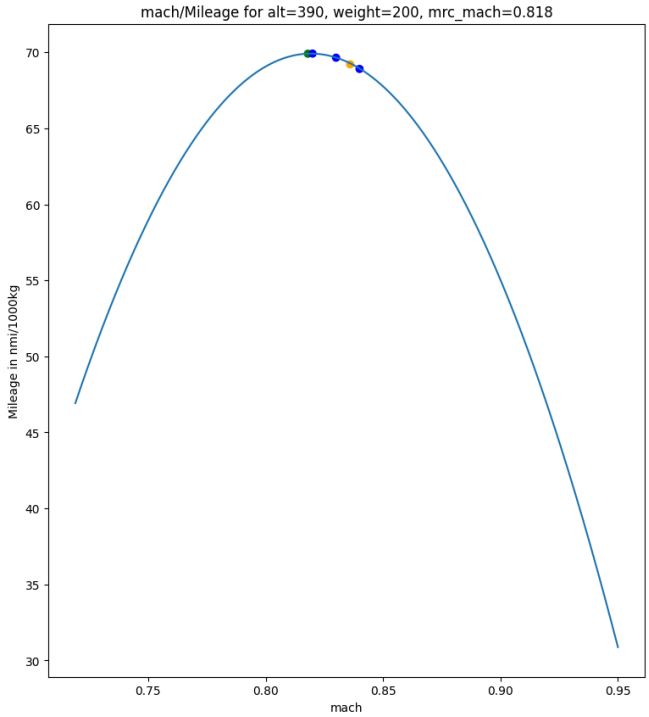
Found: 0.809 MRC and 0.835 LRC

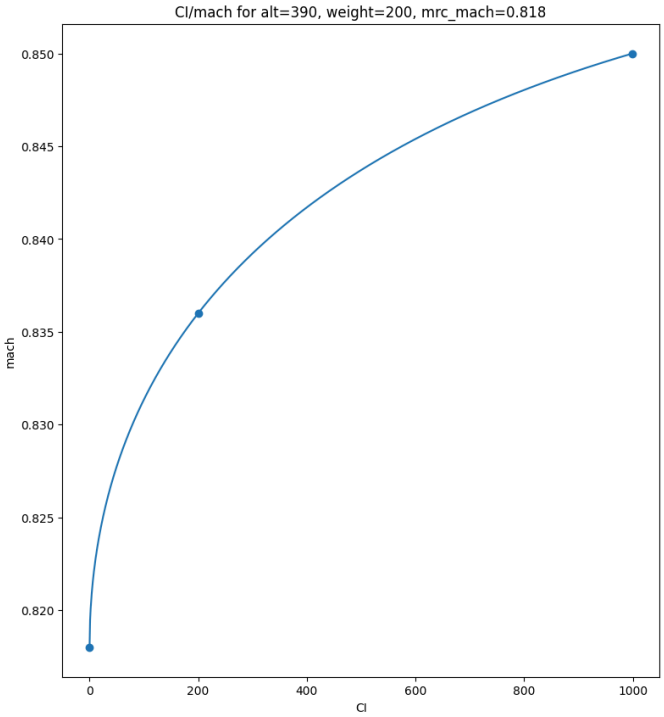




Sanity check: 0.809 MRC and 0.835 LRC

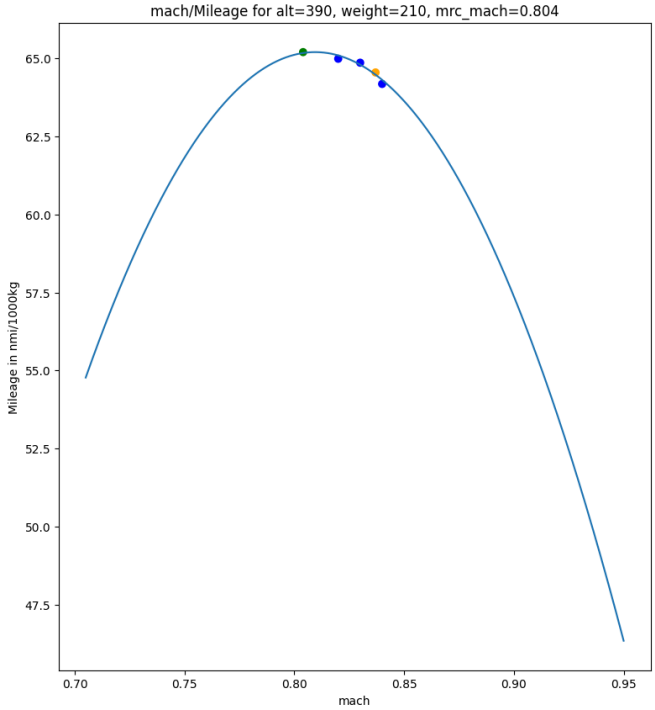
Found: 0.818 MRC and 0.836 LRC

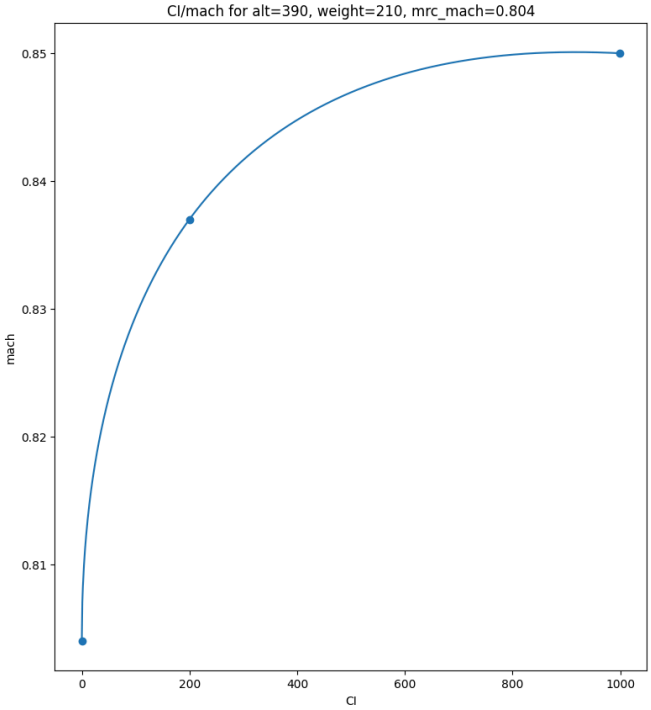




Sanity check: 0.818 MRC and 0.836 LRC

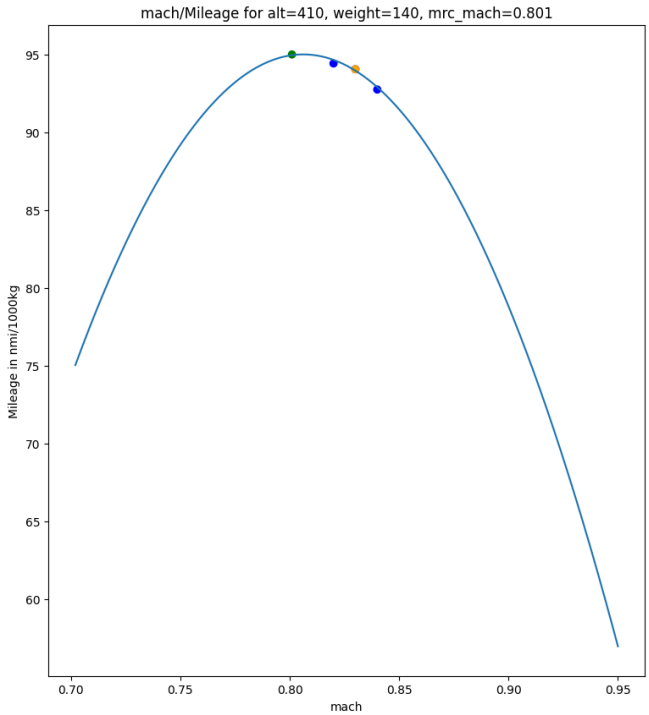
Found: 0.804 MRC and 0.837 LRC

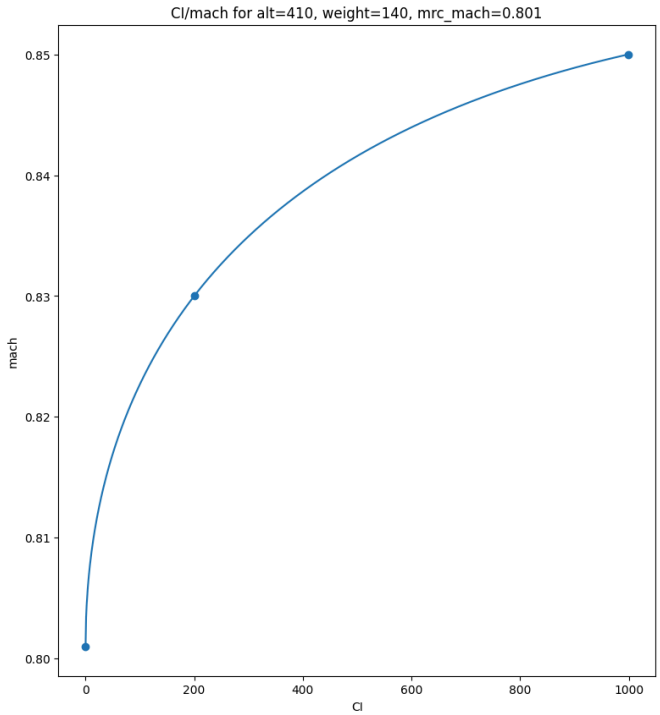




Sanity check: 0.804 MRC and 0.837 LRC

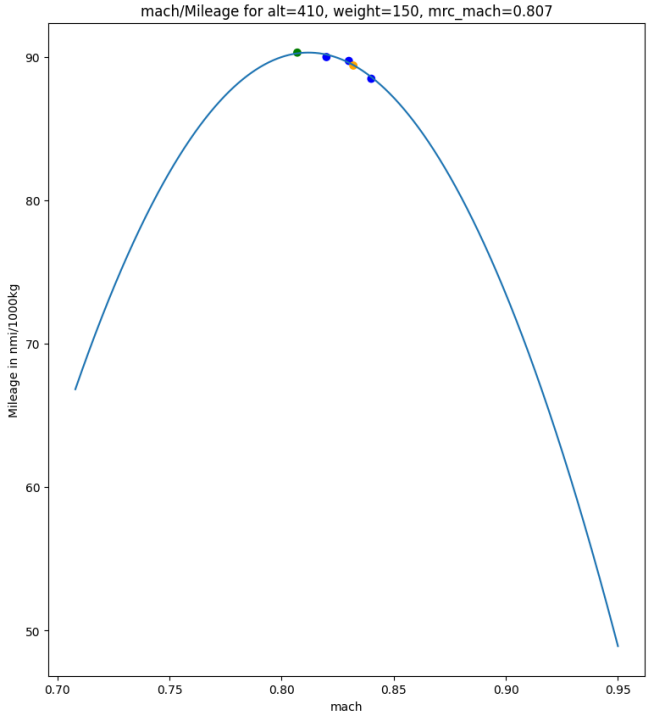
Found: 0.801 MRC and 0.83 LRC

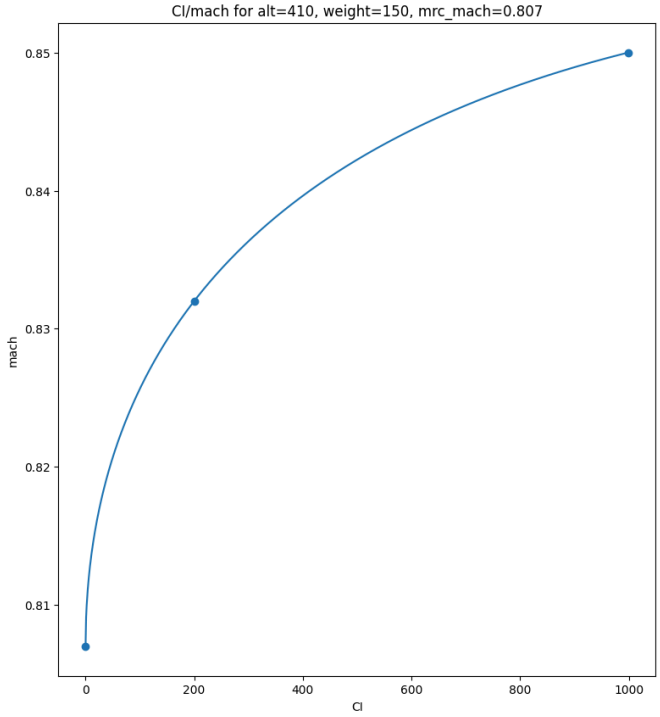




Sanity check: 0.801 MRC and 0.83 LRC

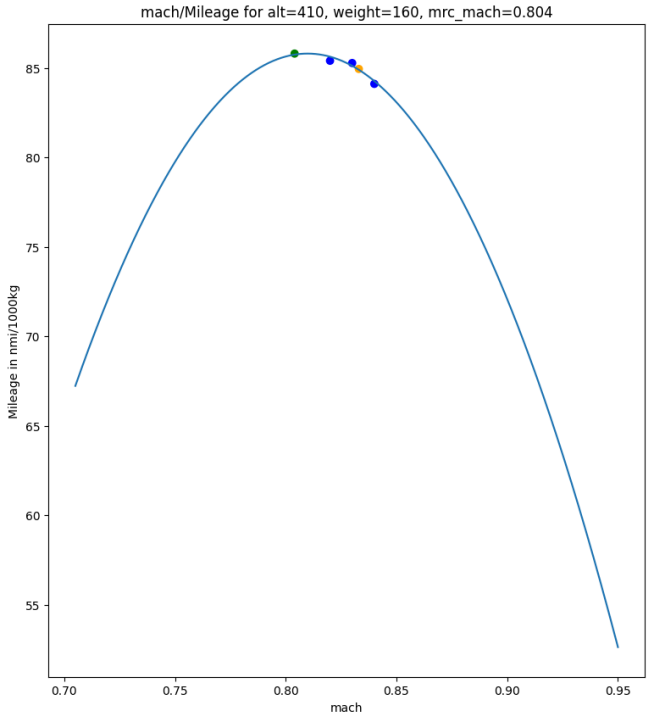
Found: 0.807 MRC and 0.832 LRC

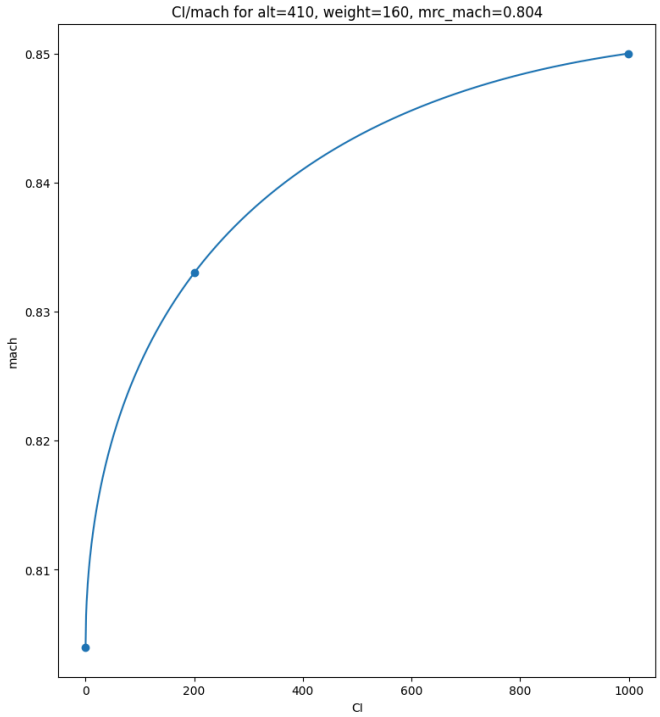




Sanity check: 0.807 MRC and 0.832 LRC

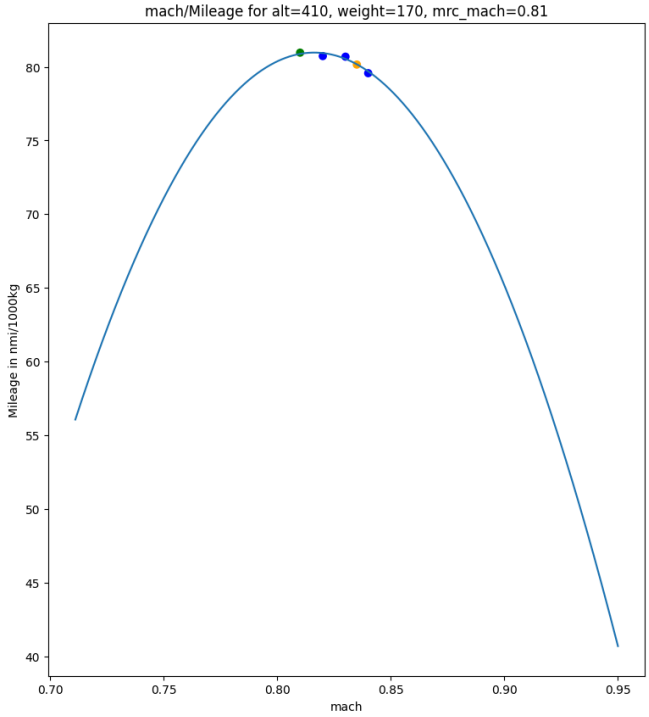
Found: 0.804 MRC and 0.833 LRC

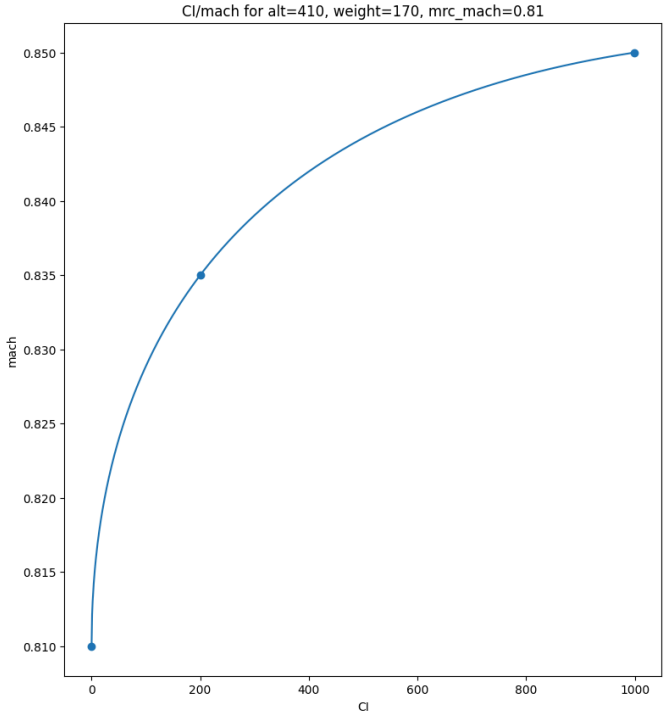




Sanity check: 0.804 MRC and 0.833 LRC

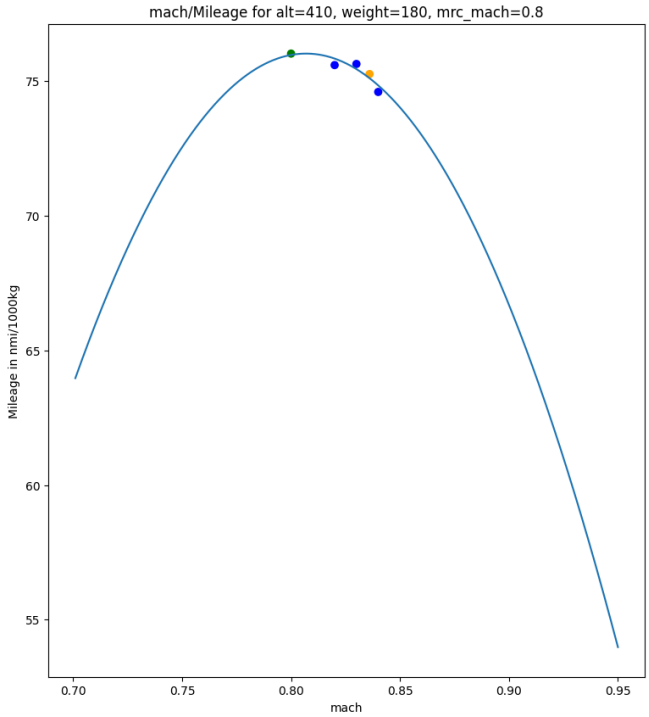
Found: 0.81 MRC and 0.835 LRC

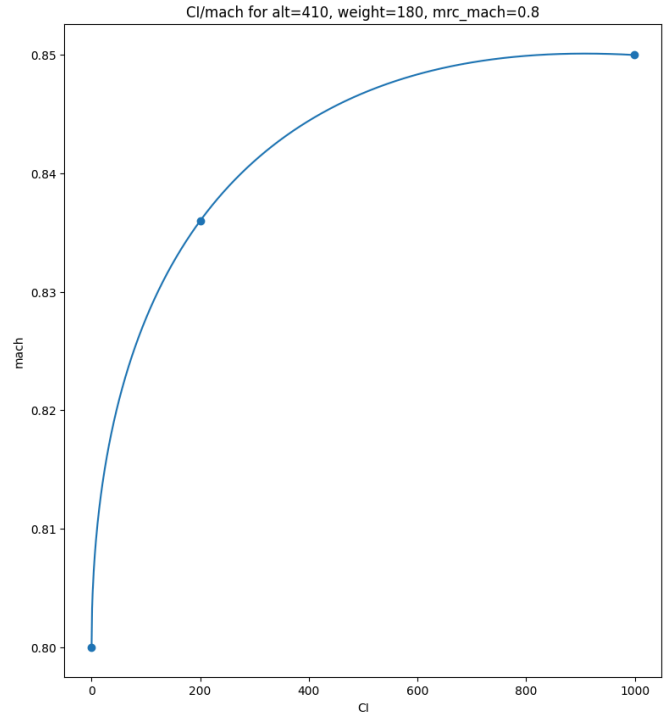




Sanity check: 0.81 MRC and 0.835 LRC

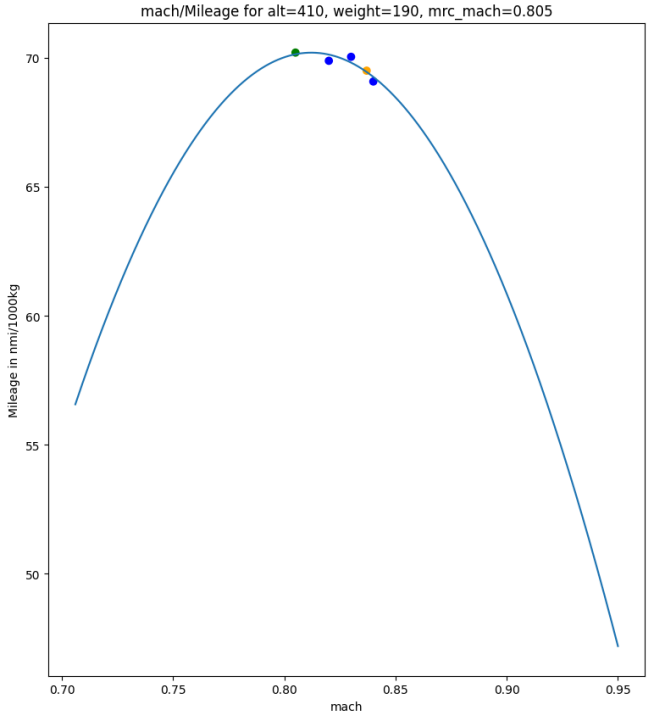
Found: 0.8 MRC and 0.836 LRC

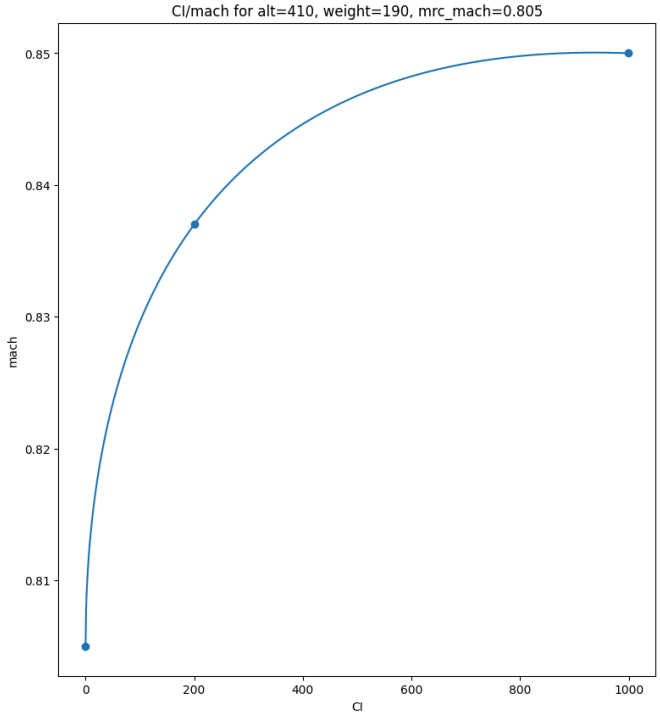




Sanity check: 0.8 MRC and 0.836 LRC

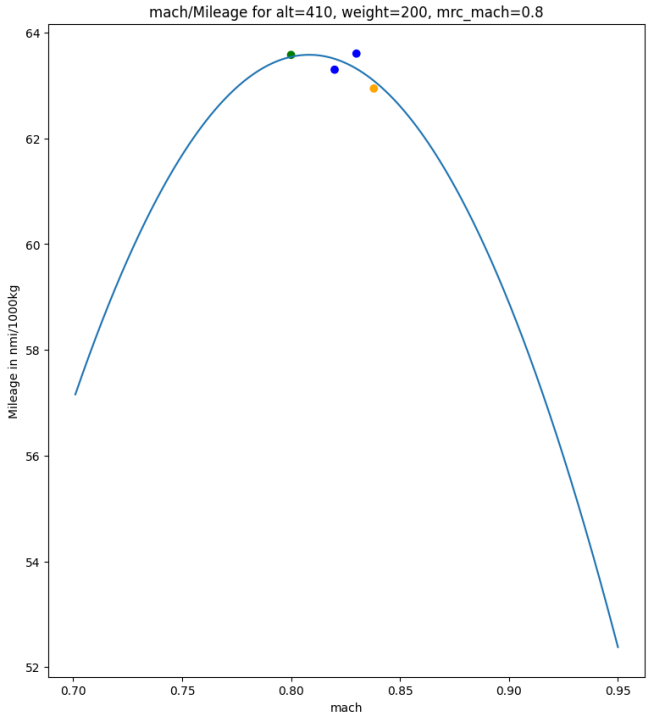
Found: 0.805 MRC and 0.837 LRC

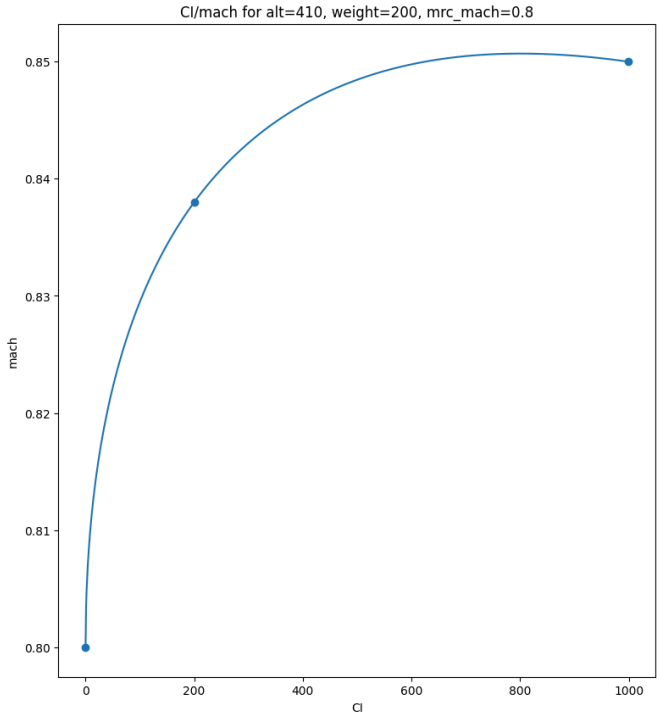




Sanity check: 0.805 MRC and 0.837 LRC

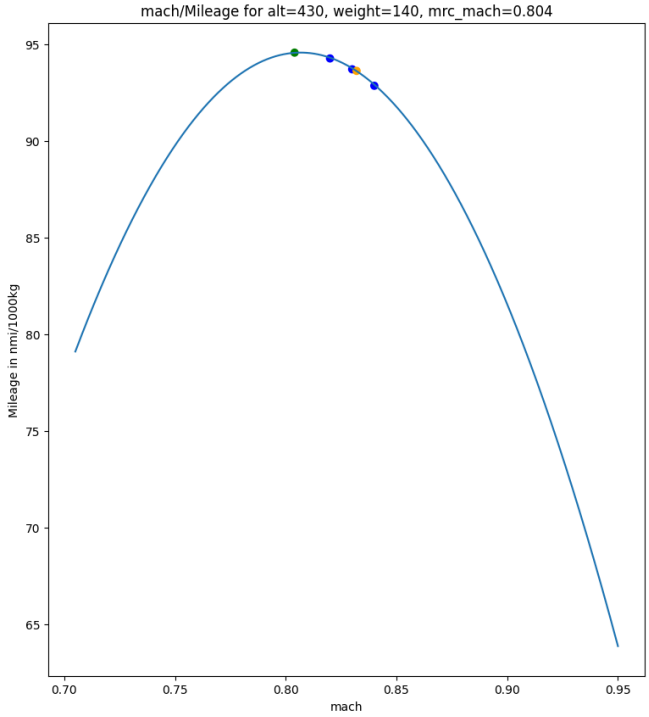
Found: 0.8 MRC and 0.838 LRC

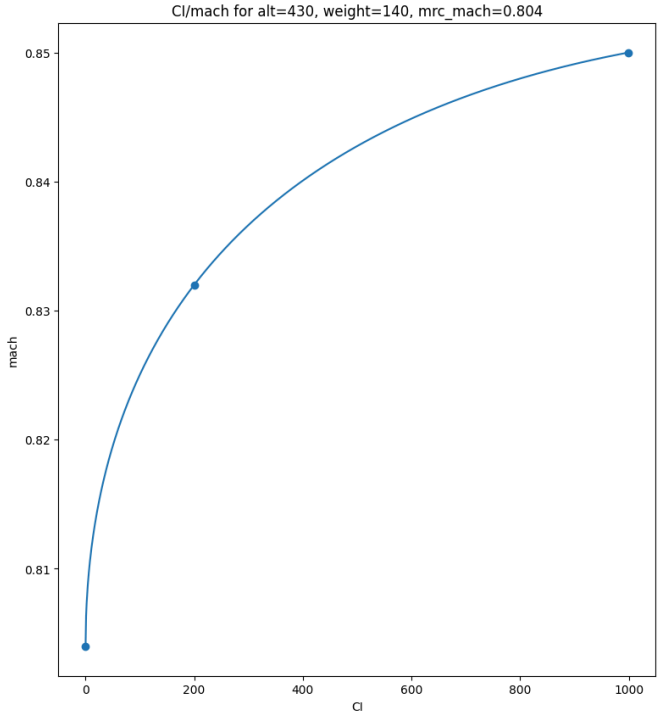




Sanity check: 0.8 MRC and 0.838 LRC

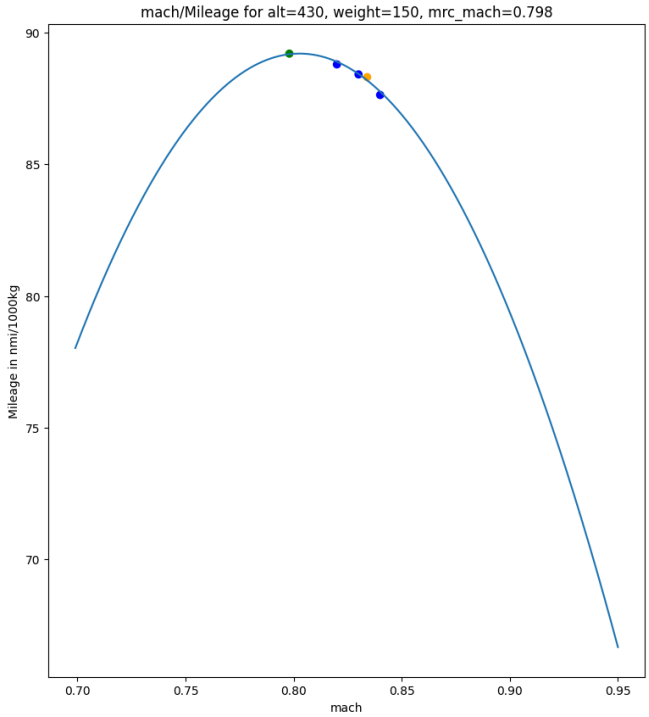
Found: 0.804 MRC and 0.832 LRC

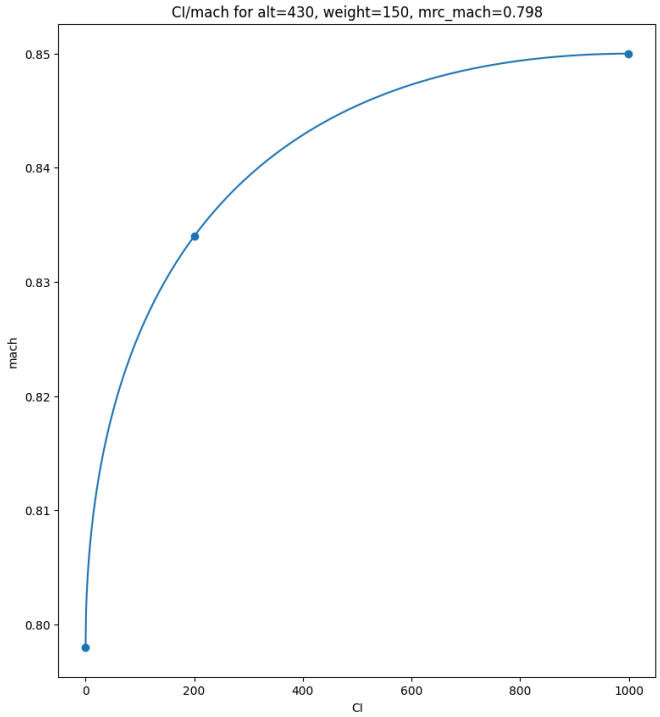




Sanity check: 0.804 MRC and 0.832 LRC

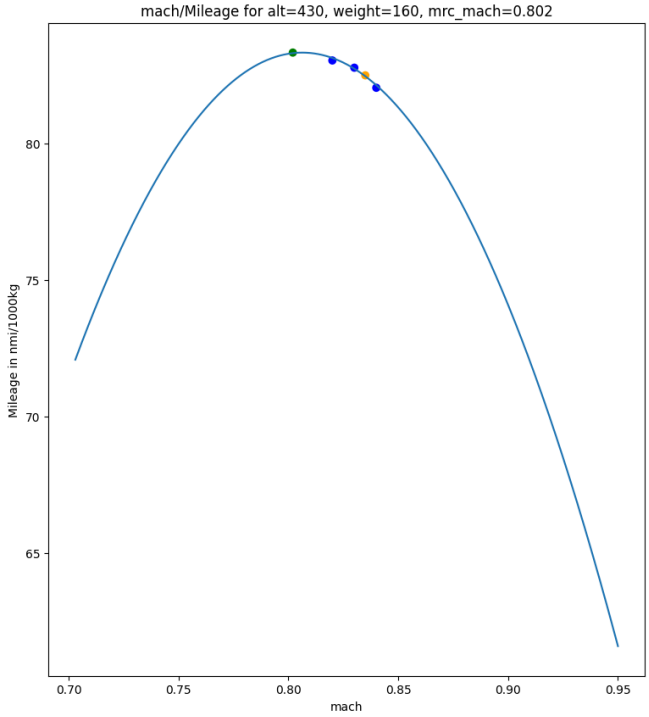
Found: 0.798 MRC and 0.834 LRC

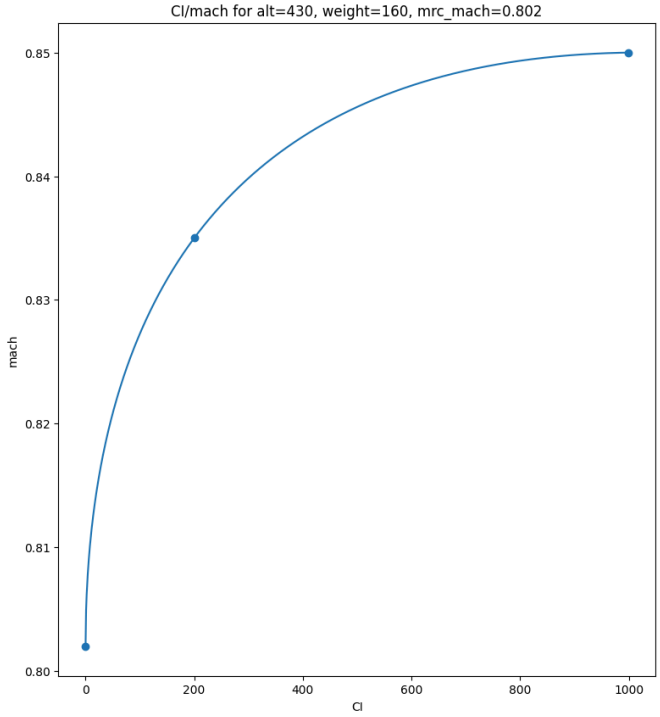




Sanity check: 0.798 MRC and 0.834 LRC

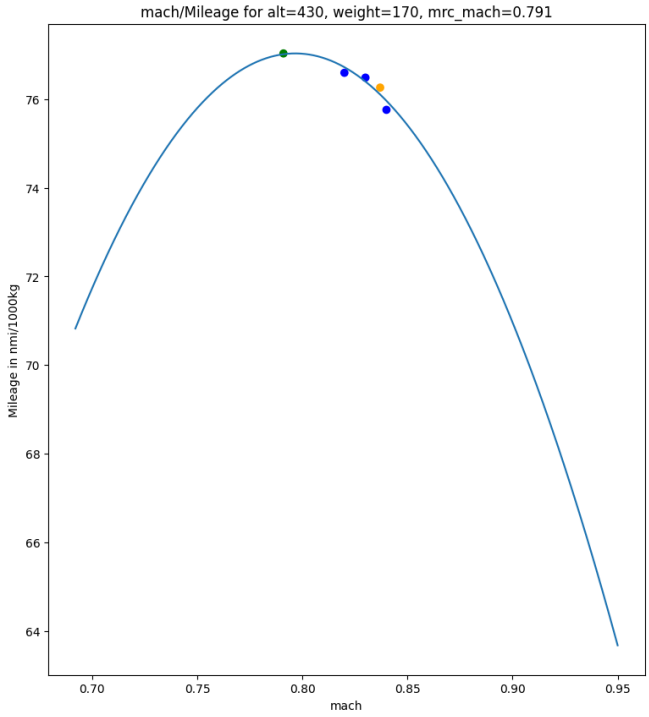
Found: 0.802 MRC and 0.835 LRC

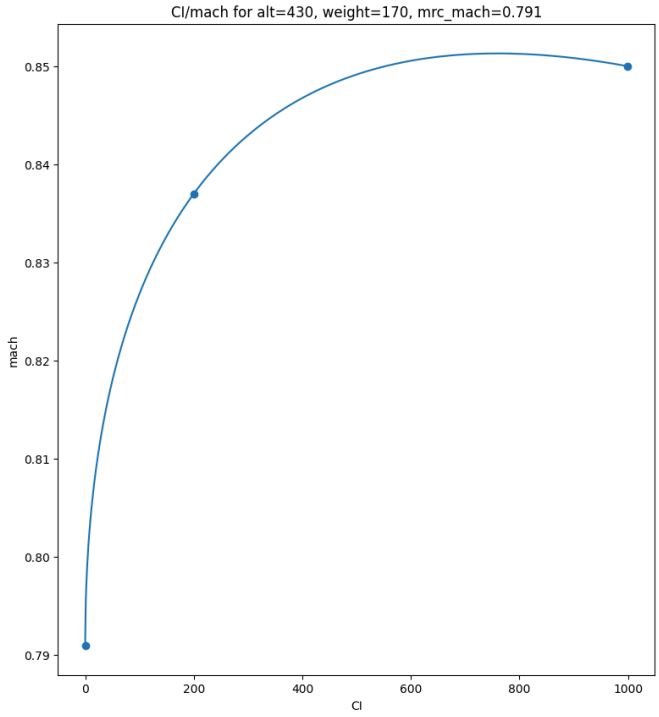




Sanity check: 0.802 MRC and 0.835 LRC

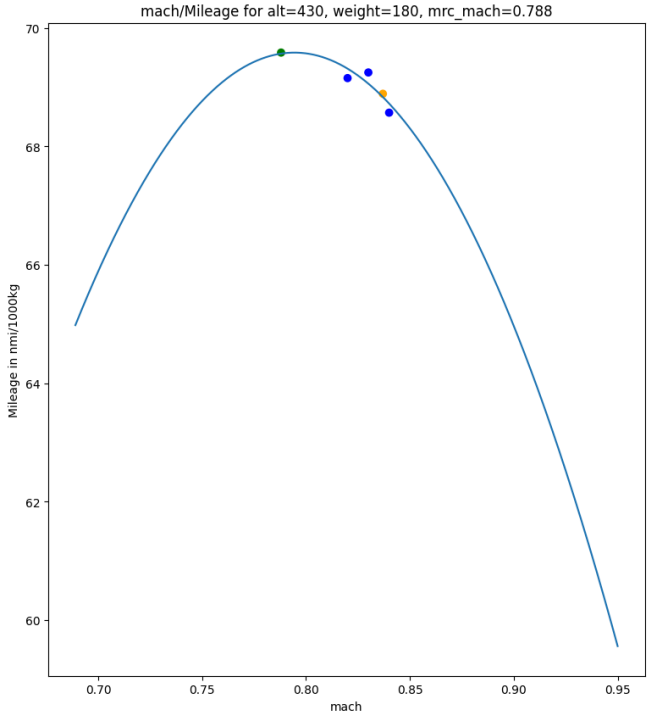
Found: 0.791 MRC and 0.837 LRC

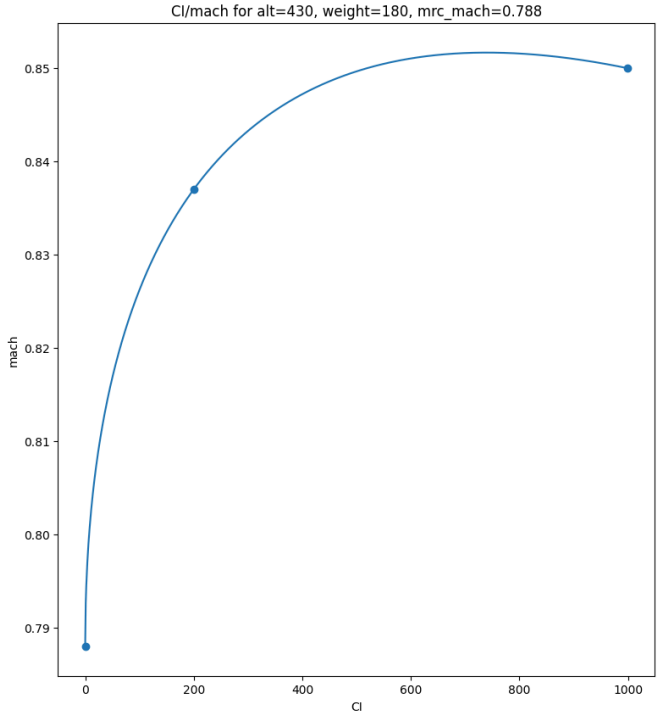




Sanity check: 0.791 MRC and 0.837 LRC

Found: 0.788 MRC and 0.837 LRC

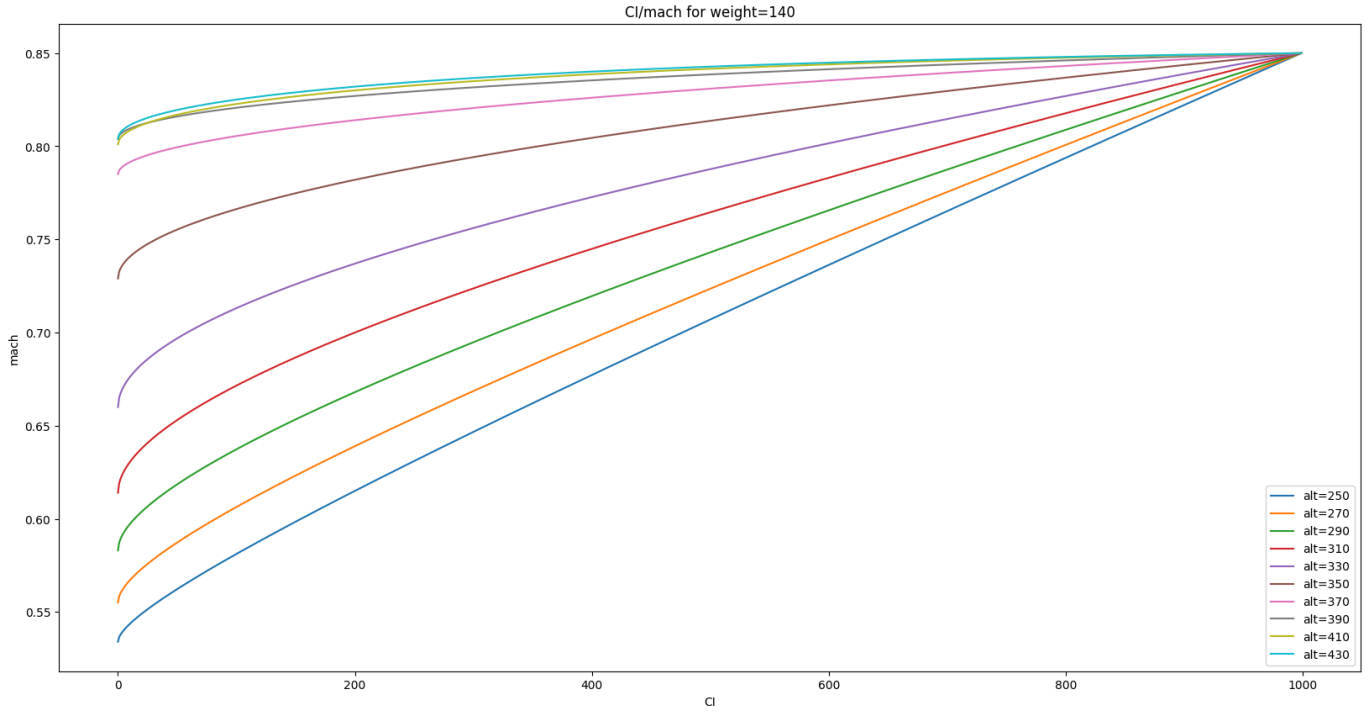


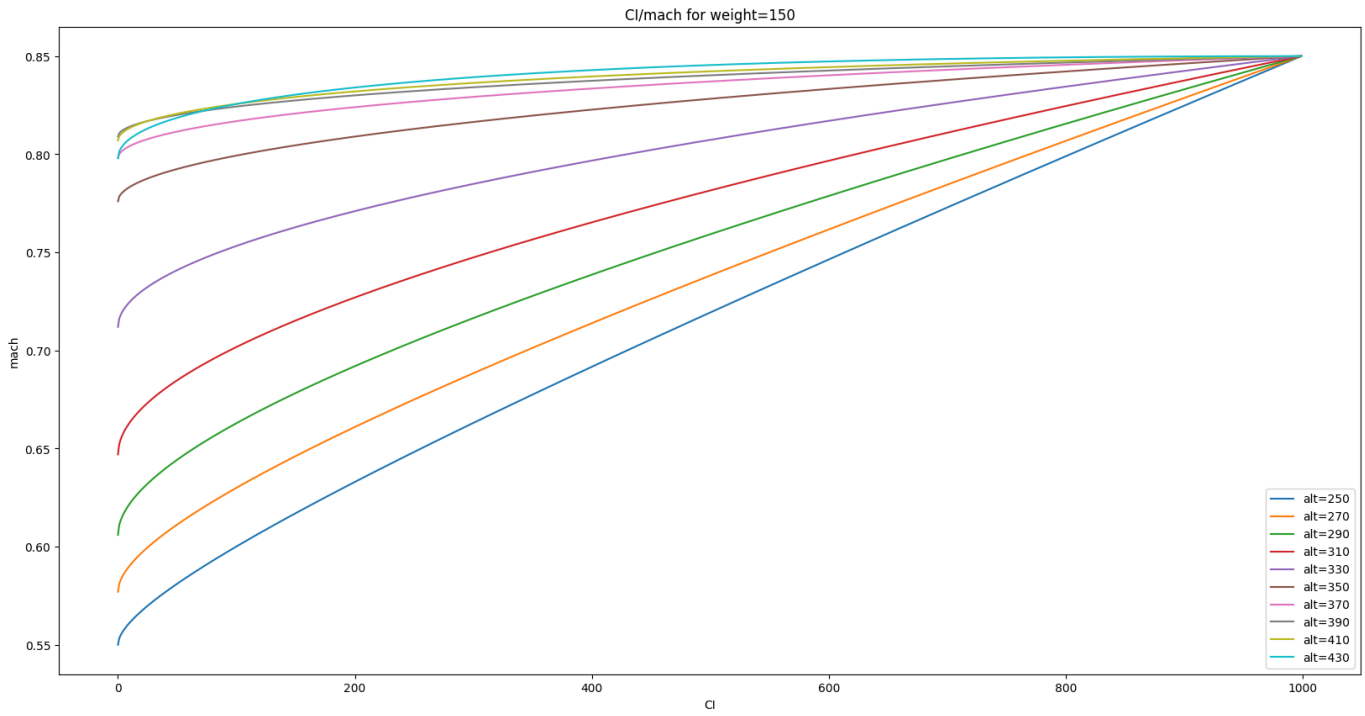


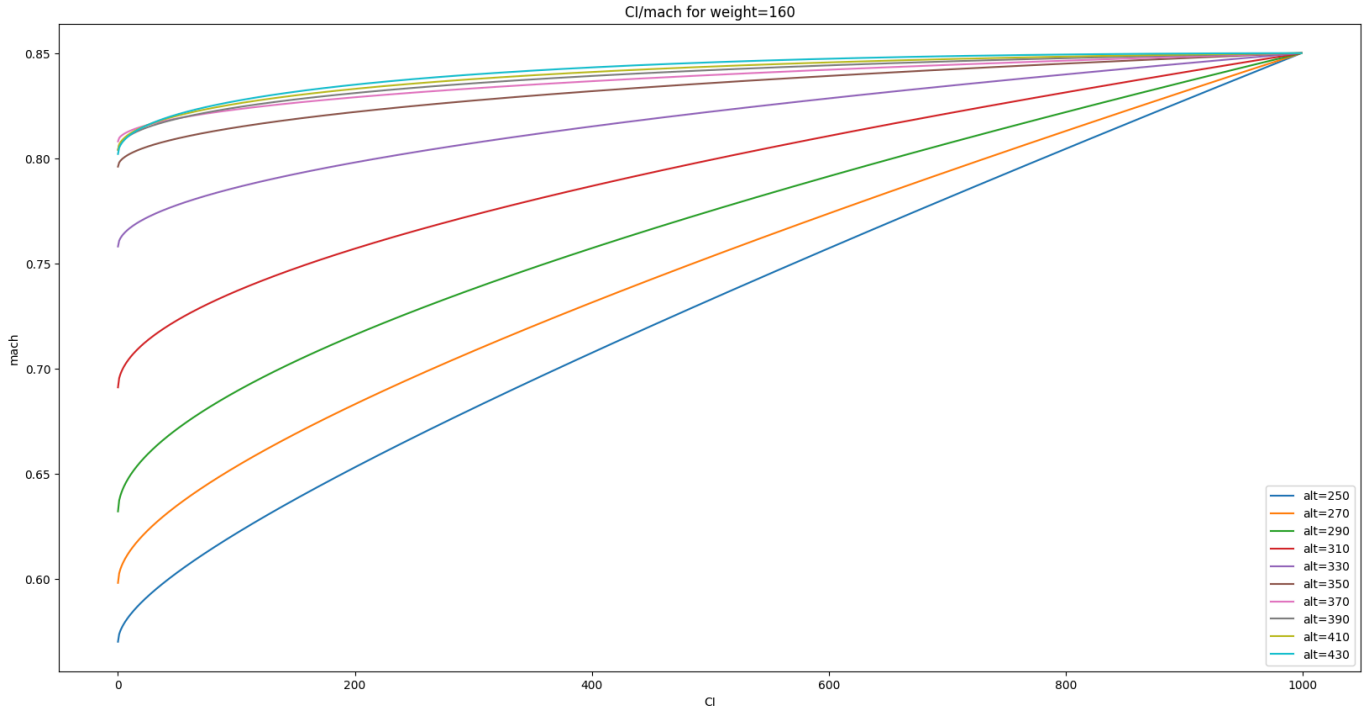
Sanity check: 0.788 MRC and 0.837 LRC

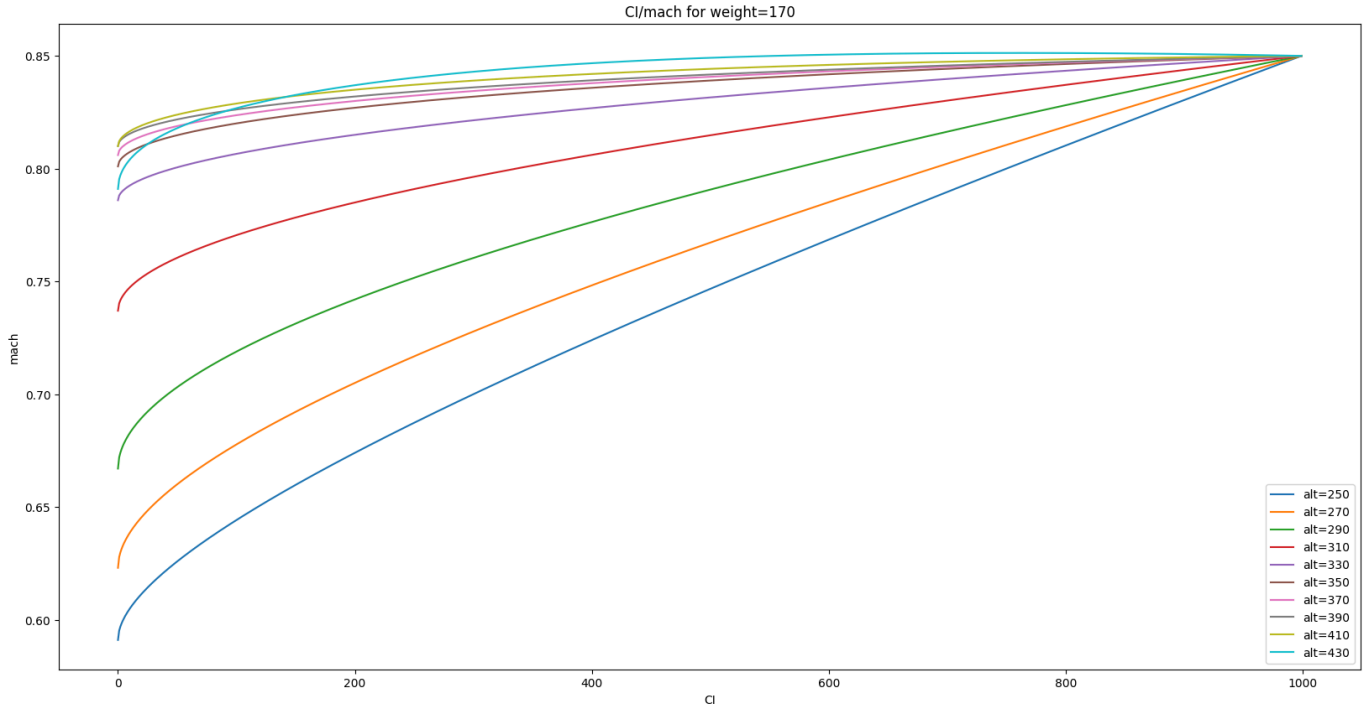
Diagnostic plots to see how the CI/mach curves change with altitude for a given weight.

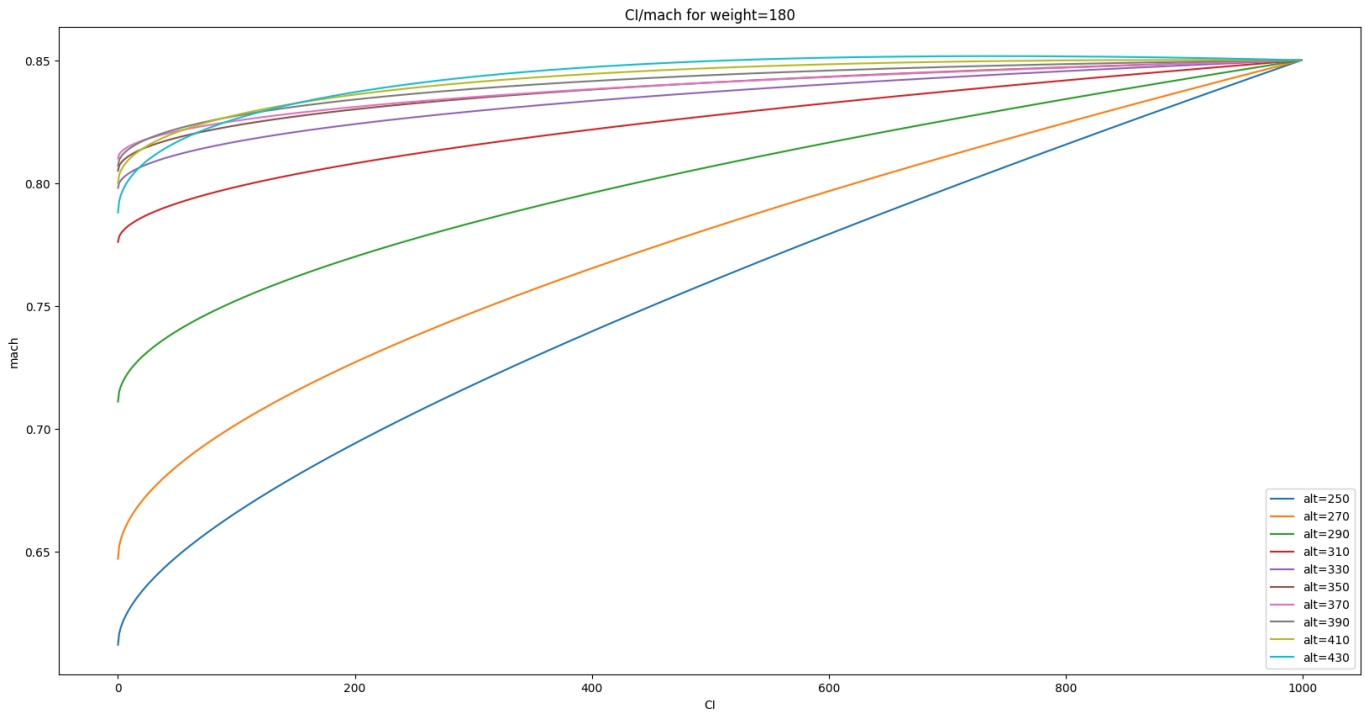
In [ ]:
```
if not skip_plots:
  x = np.linspace(0, 999, points_for_fit)

  for weight in weights_index:
    plt.figure(figsize=(20,10))
    plt.title(f"CI/mach for weight={index2weight(weight)}")
    plt.xlabel("CI")
    plt.ylabel("mach")
    for alt in alts_index:
      try:
        plt.plot(x, curves[alt][weight], label=f"alt={index2alt(alt)}")
      except IndexError:
        pass
    plt.legend()
    plt.show()
```

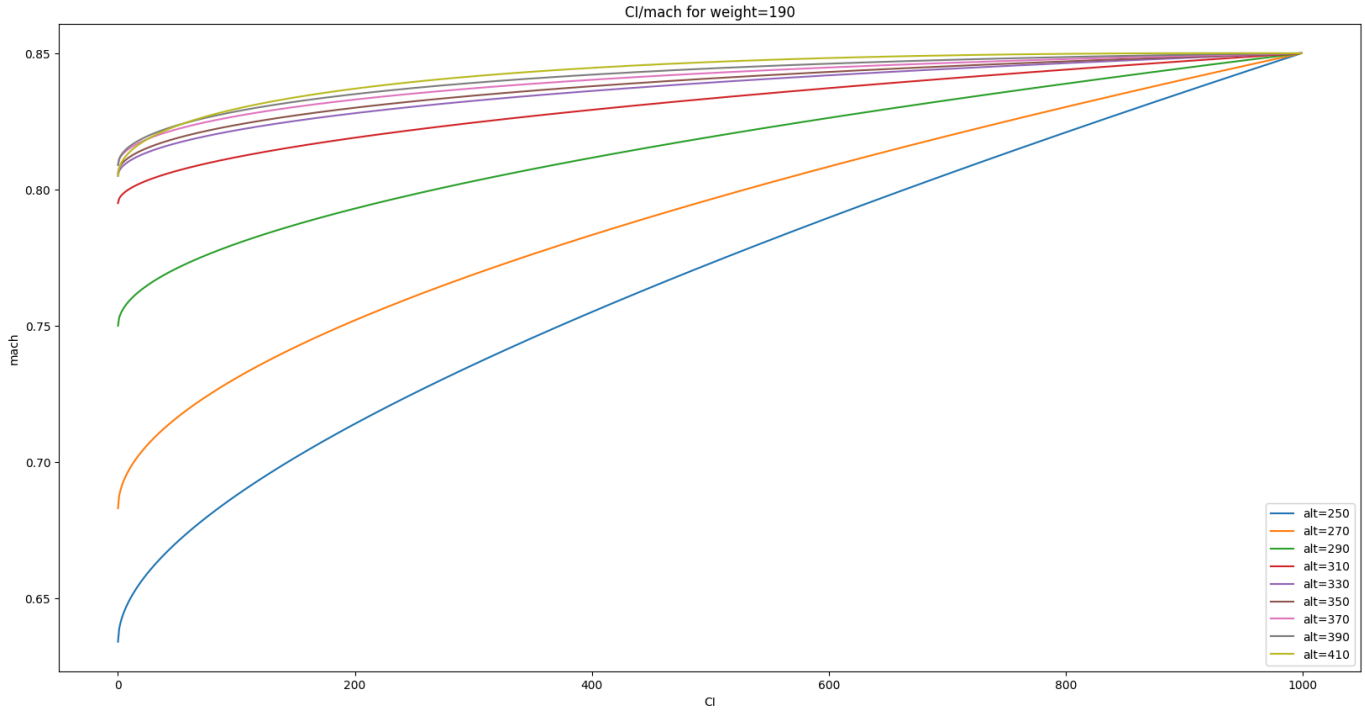


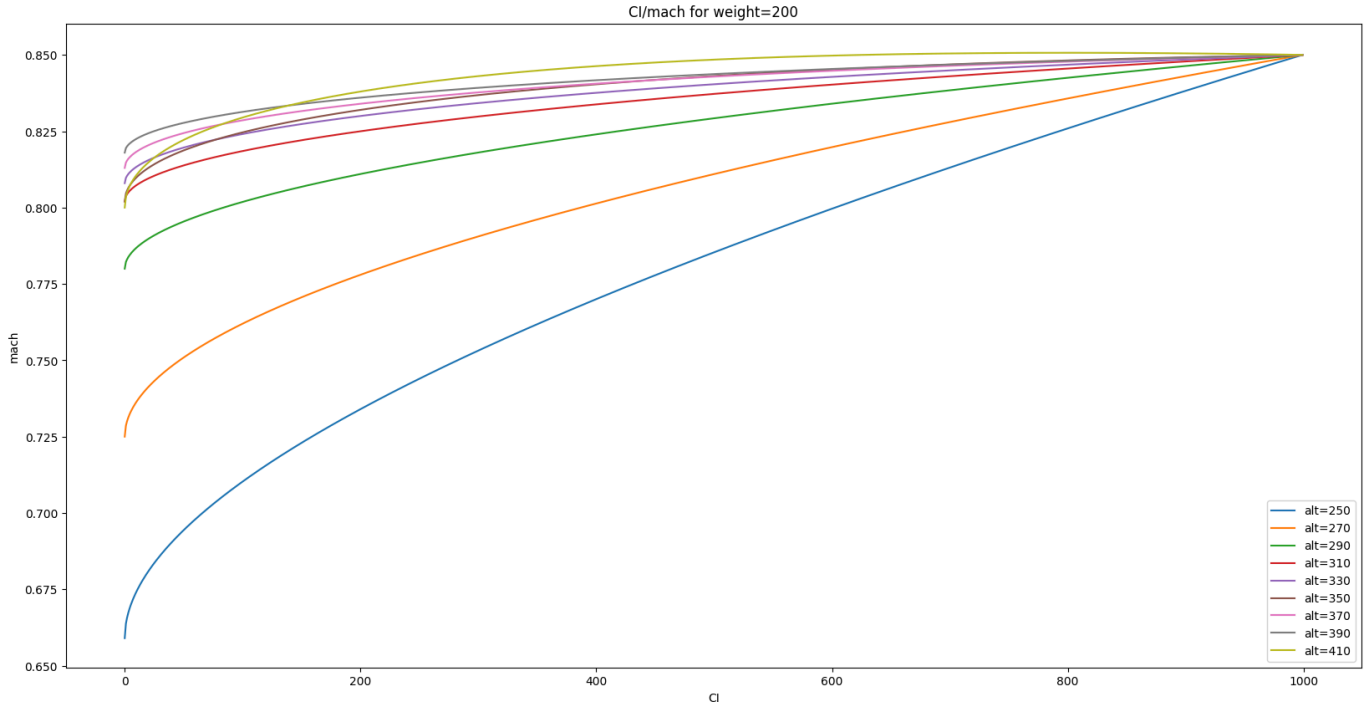

Cl/mach for weight=150
mach
Cl
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
200
400
600
800
1000
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410
alt=430

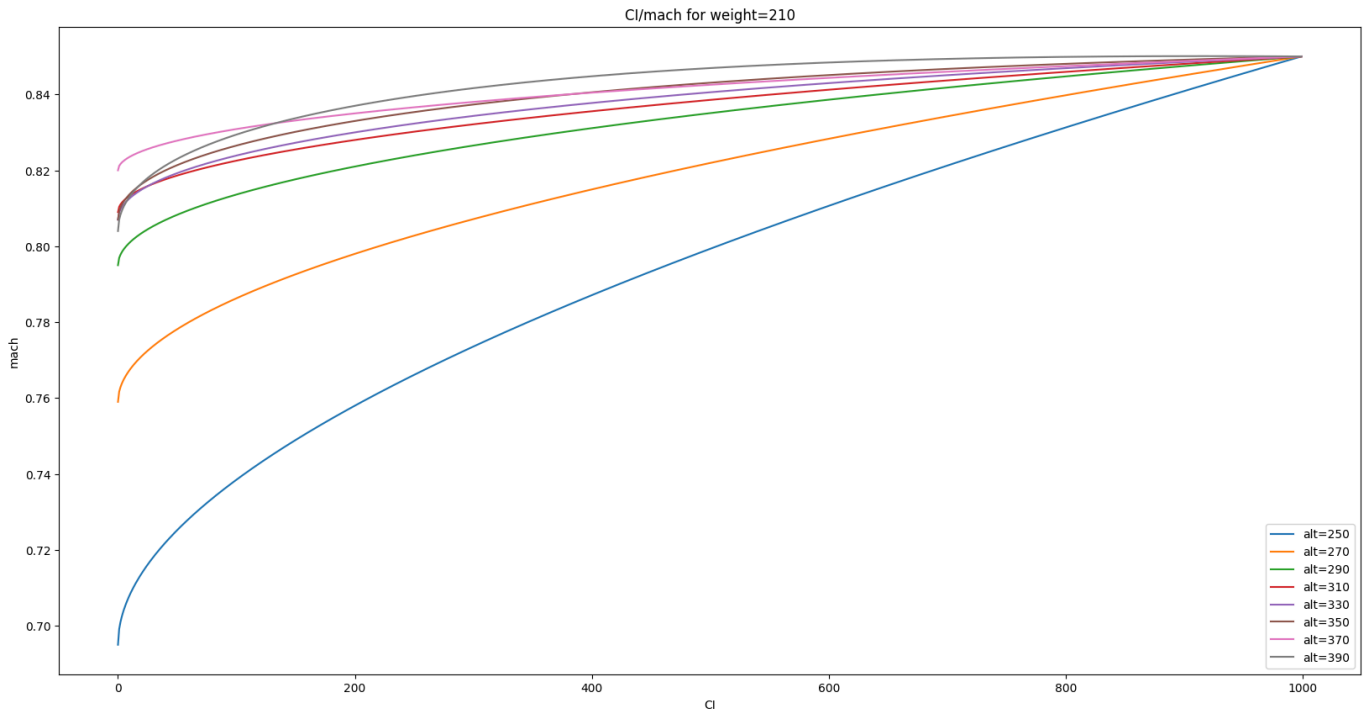

Cl/mach for weight=160
mach
Cl
0.85
0.80
0.75
0.70
0.65
0.60
0
200
400
600
800
1000
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410
alt=430

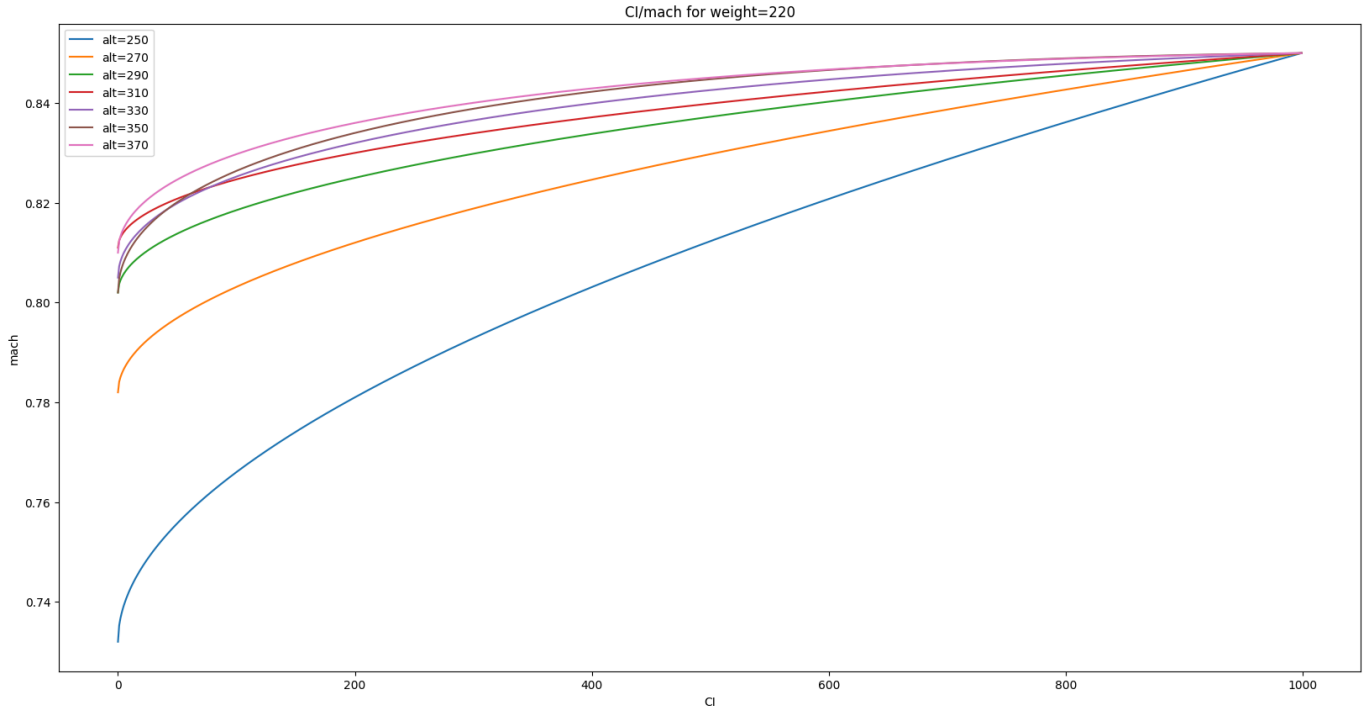

Cl/mach for weight=170
mach
Cl
0.85
0.80
0.75
0.70
0.65
0.60
0
200
400
600
800
1000
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410
alt=430

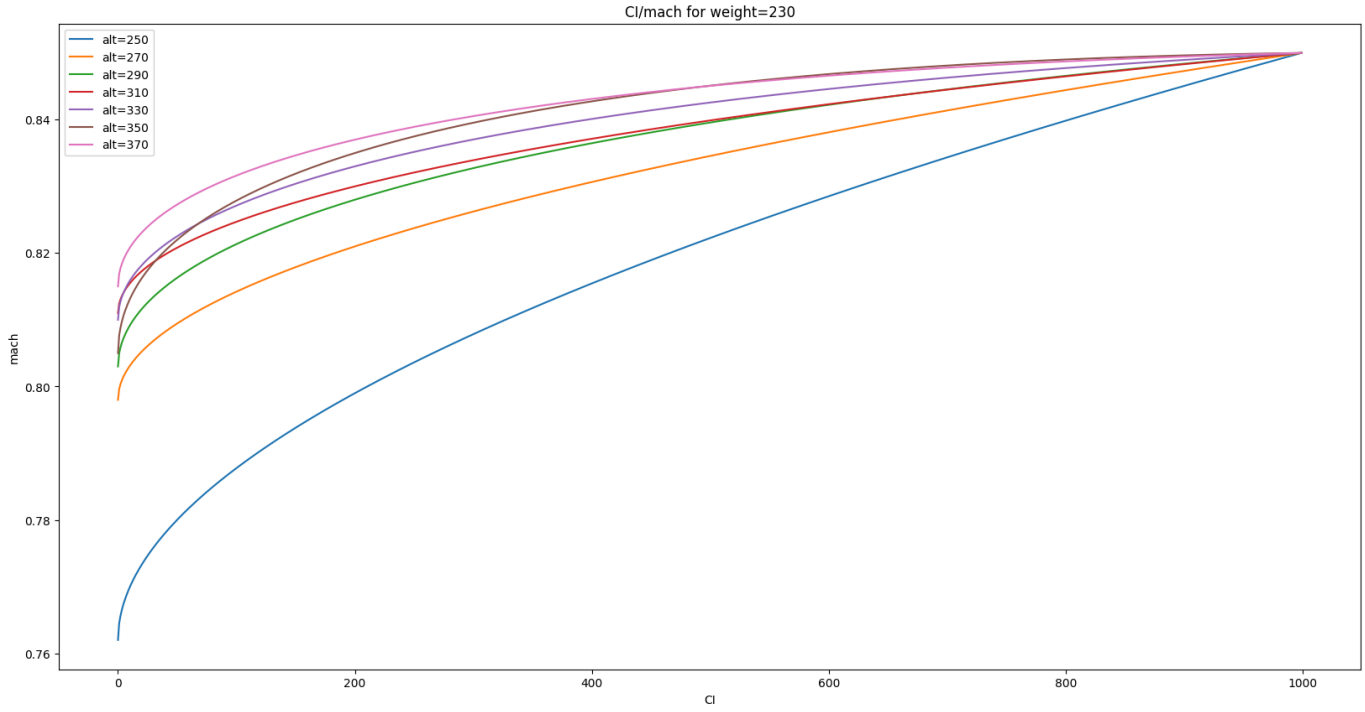

Cl/mach for weight=180
mach
Cl
0.85
0.80
0.75
0.70
0.65
0
200
400
600
800
1000
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410
alt=430

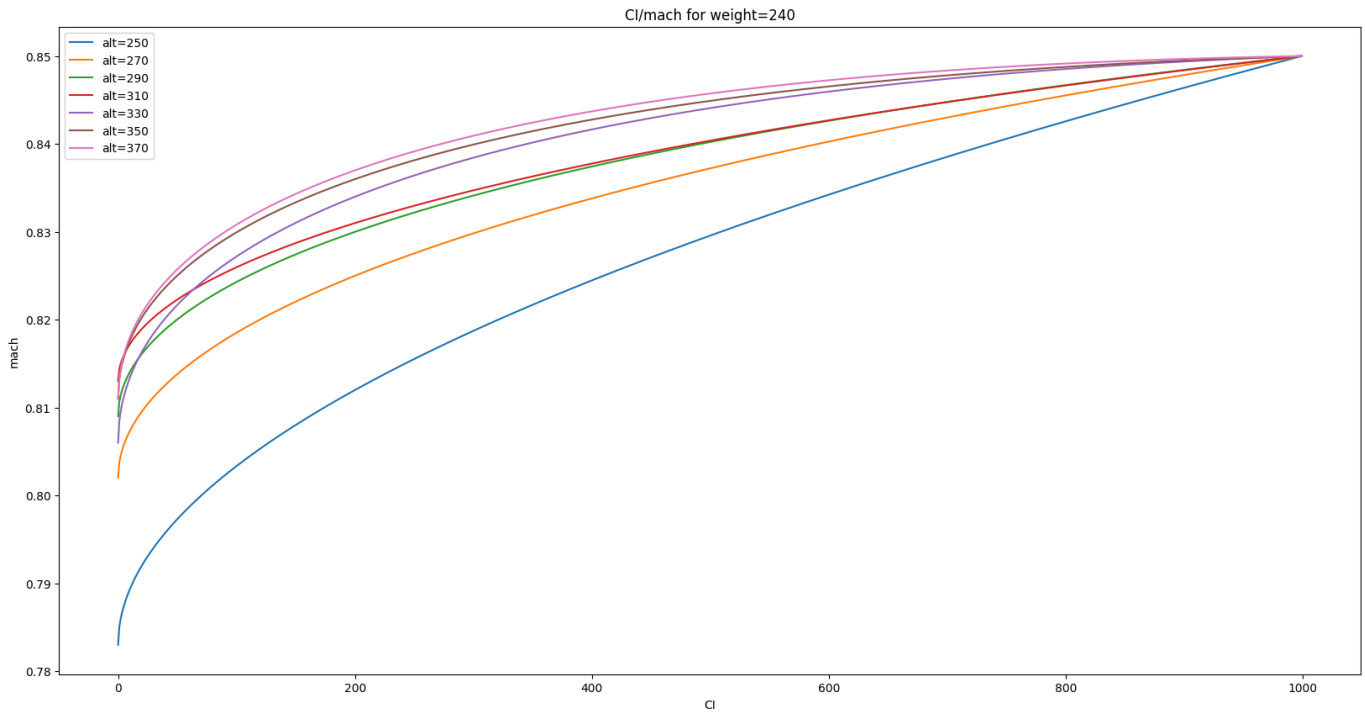

Cl/mach for weight=190
mach
Cl
0.85
0.80
0.75
0.70
0.65
0
200
400
600
800
1000
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410

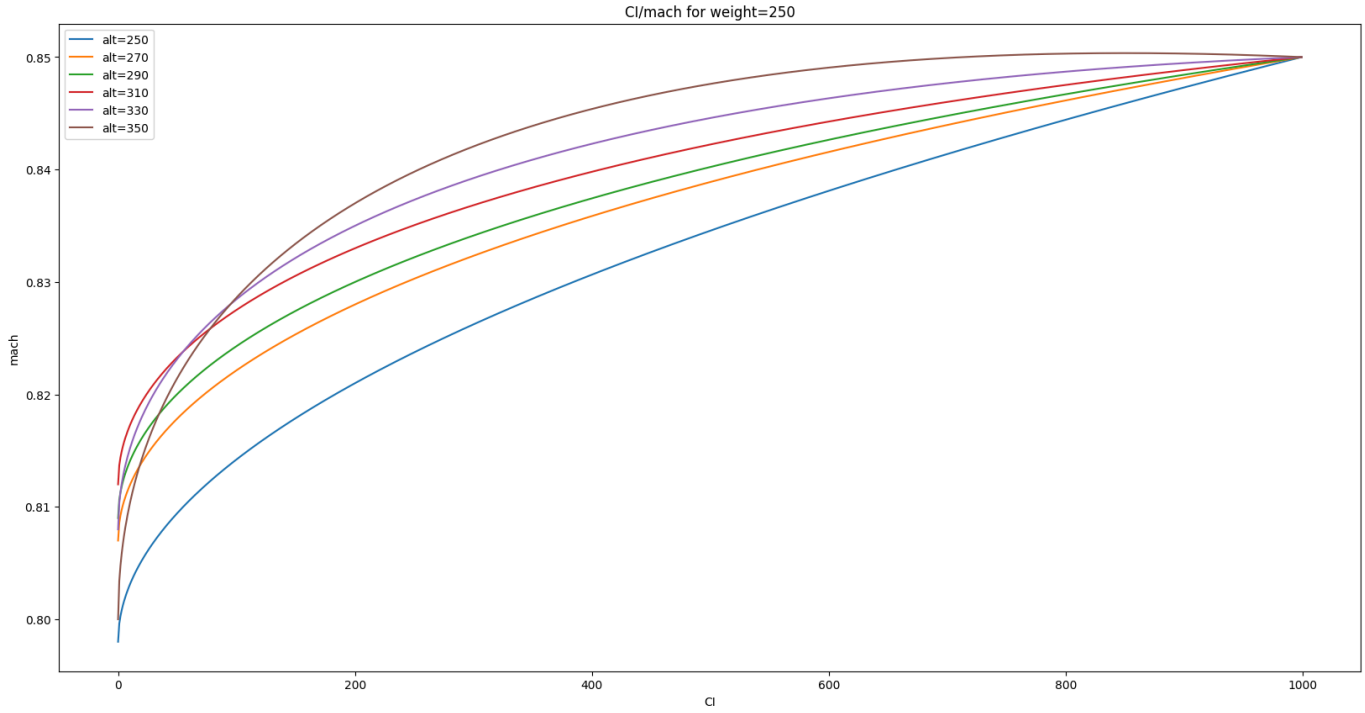

Cl/mach for weight=200
mach
Cl
0.850
0.825
0.800
0.775
0.750
0.725
0.700
0.675
0.650
0
200
400
600
800
1000
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410

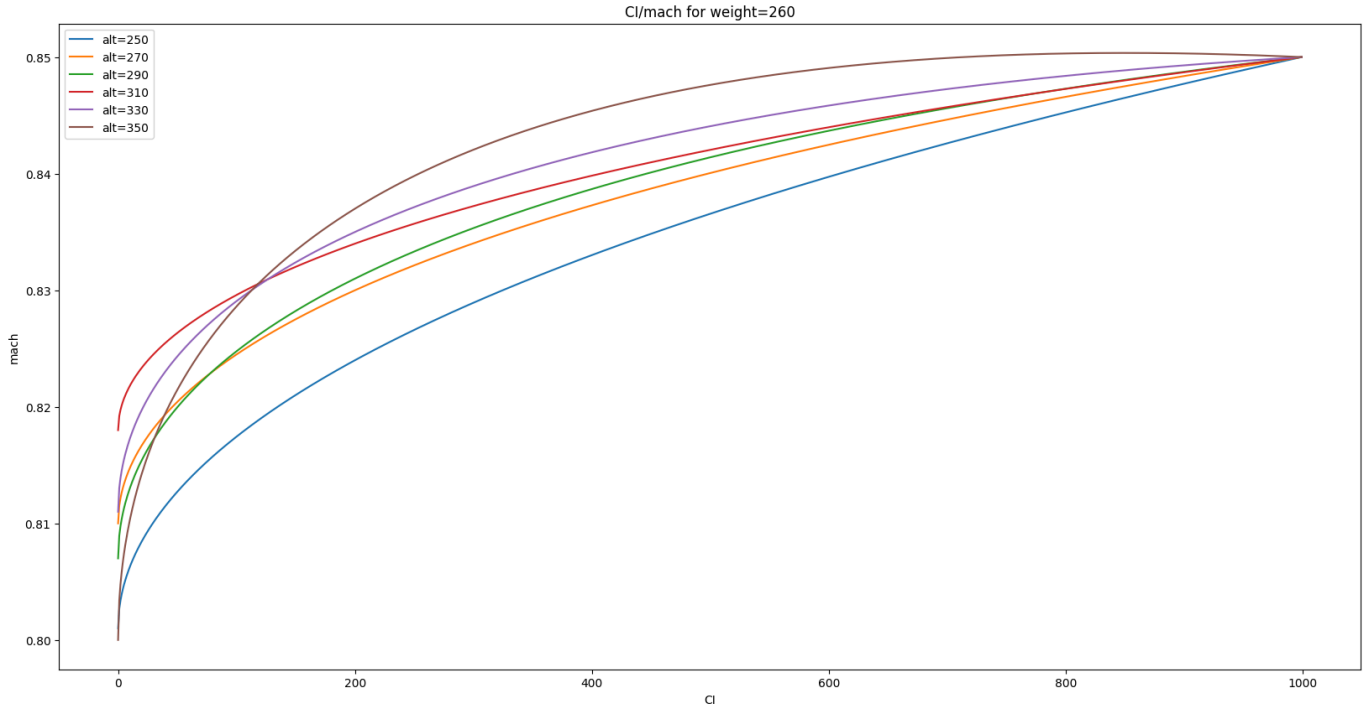

Cl/mach for weight=210
mach
Cl
0.84
0.82
0.80
0.78
0.76
0.74
0.72
0.70
0
200
400
600
800
1000
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390

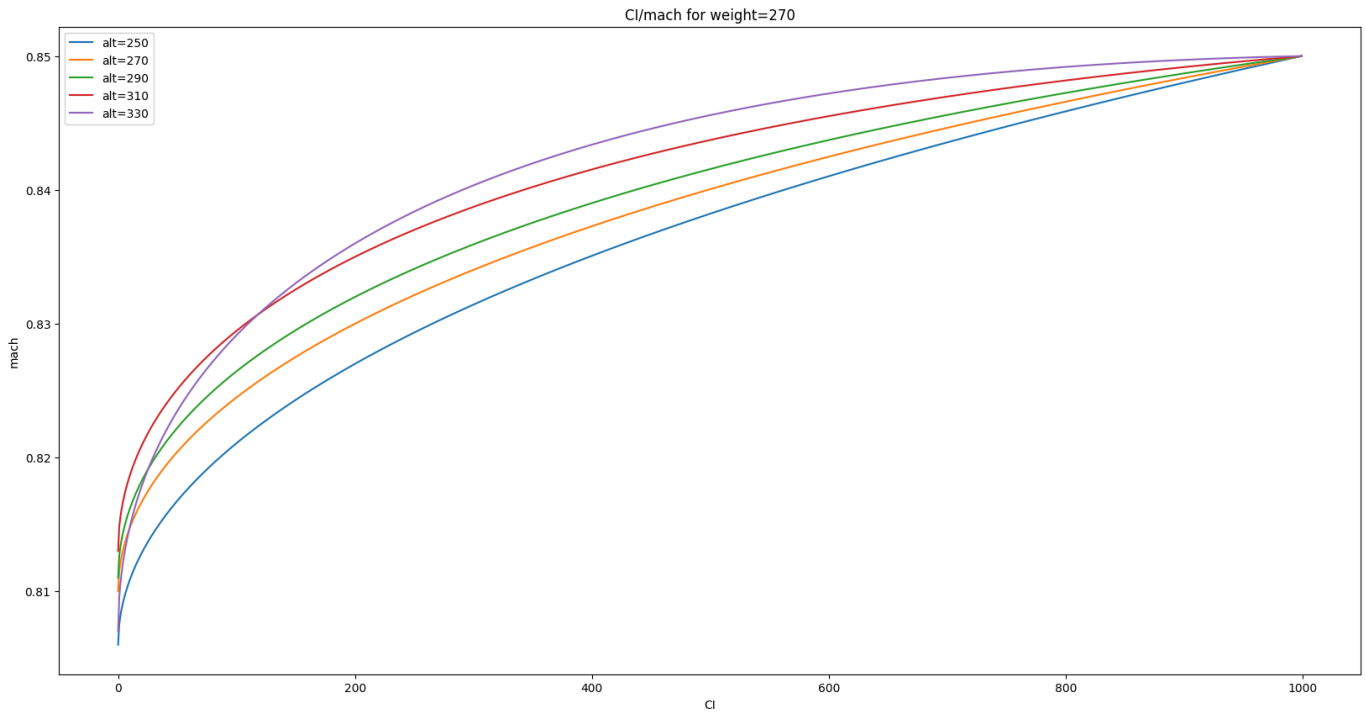

Cl/mach for weight=220
mach
Cl
0.84
0.82
0.80
0.78
0.76
0.74
0
200
400
600
800
1000
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370

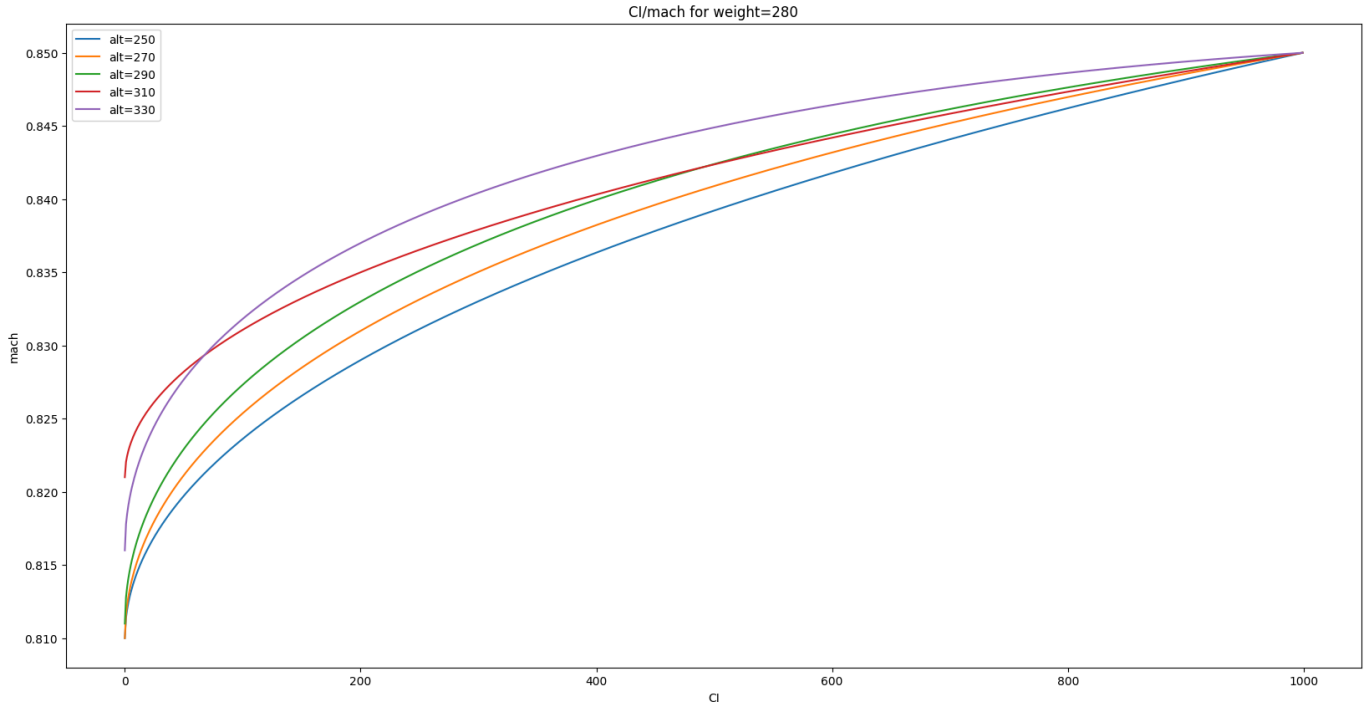

Cl/mach for weight=230
mach
Cl
0.84
0.82
0.80
0.78
0.76
0
200
400
600
800
1000
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370

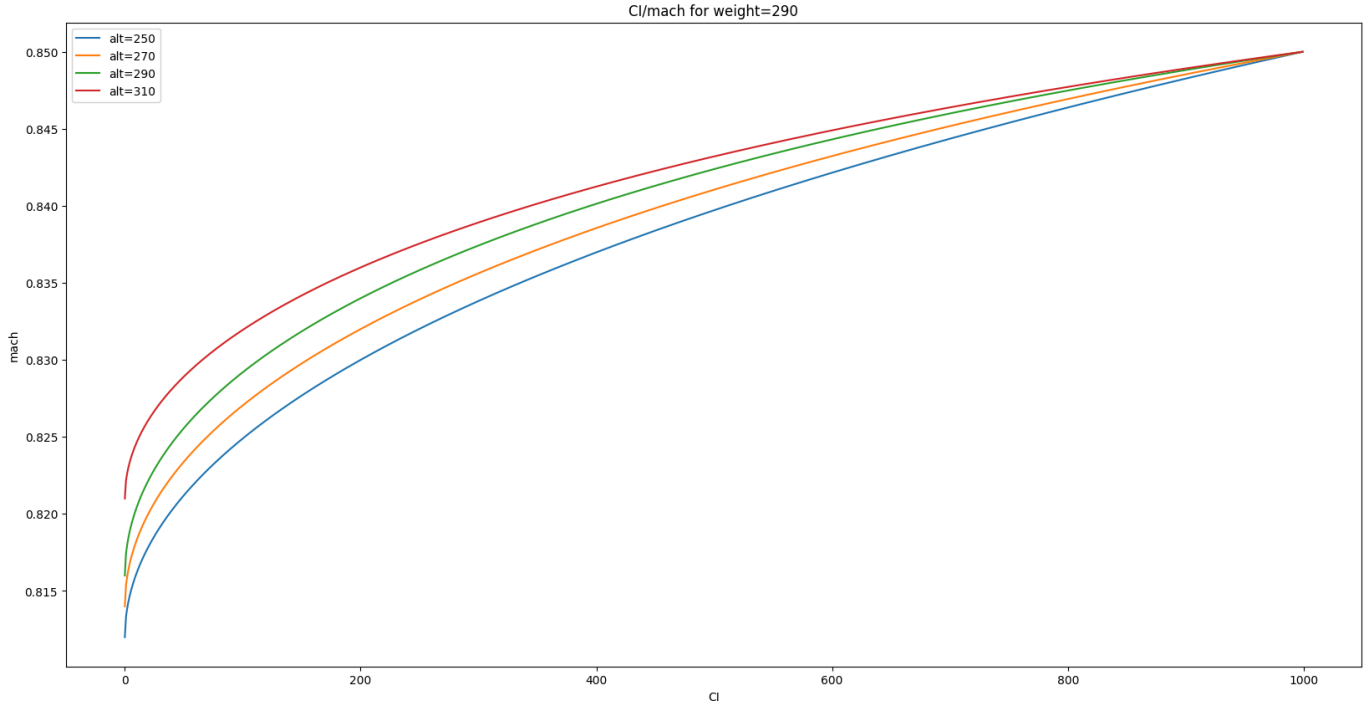

Cl/mach for weight=240
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
mach
Cl
0.85
0.84
0.83
0.82
0.81
0.80
0.79
0.78
0
200
400
600
800
1000


Cl/mach for weight=250
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
mach
Cl
0.85
0.84
0.83
0.82
0.81
0.80
0
200
400
600
800
1000


Cl/mach for weight=260
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
mach
Cl
0.85
0.84
0.83
0.82
0.81
0.80
0
200
400
600
800
1000



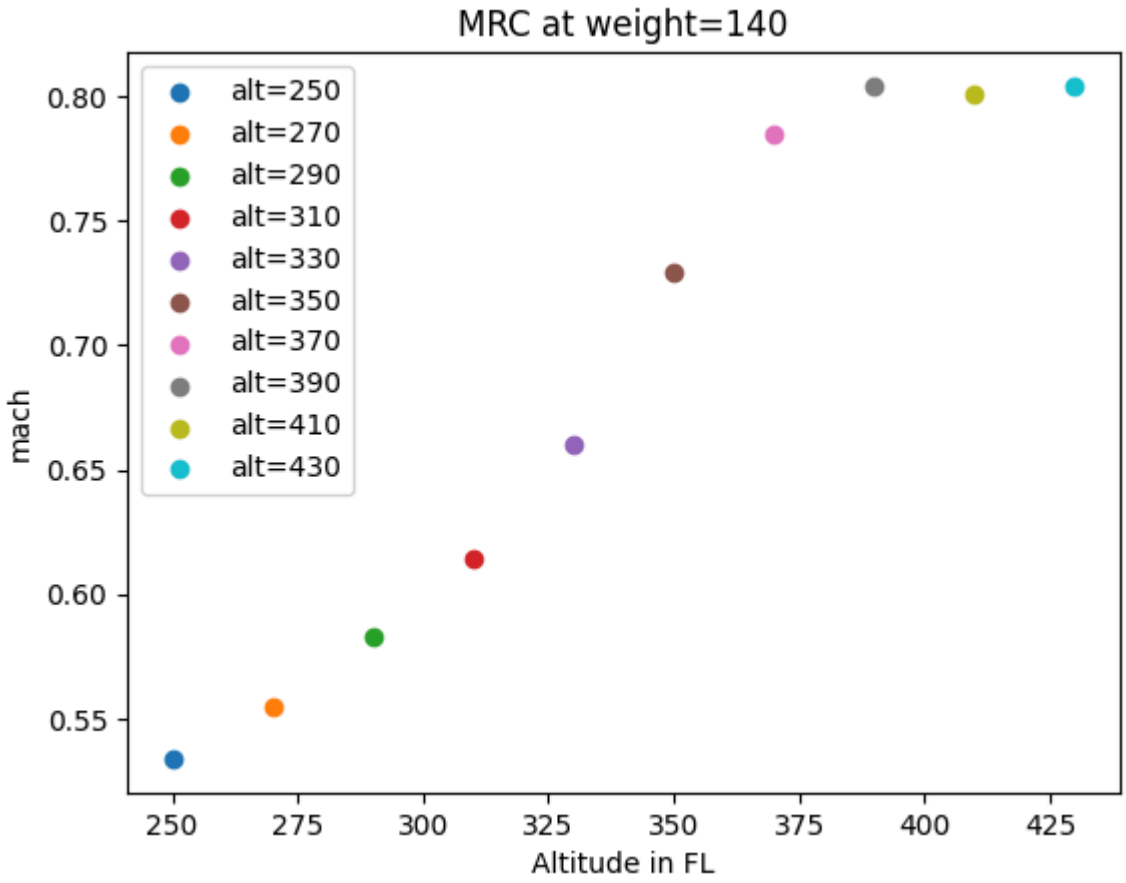


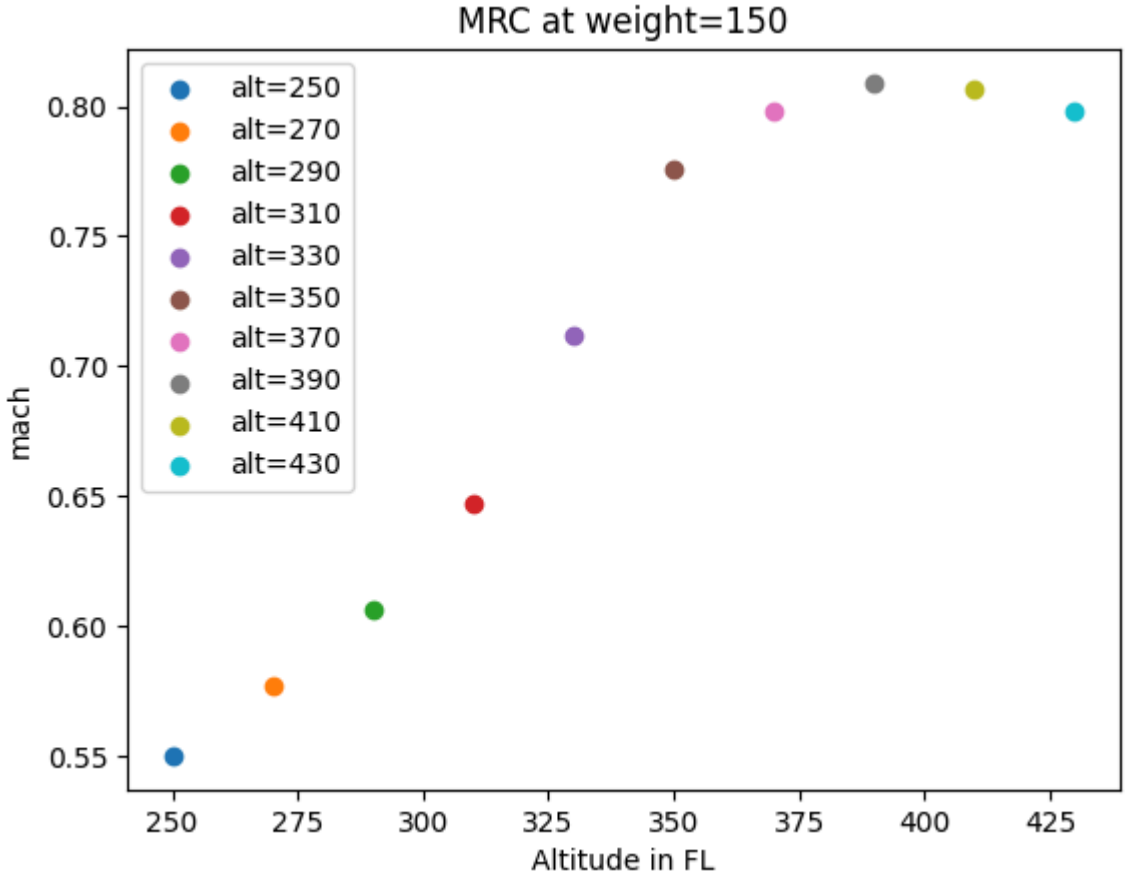


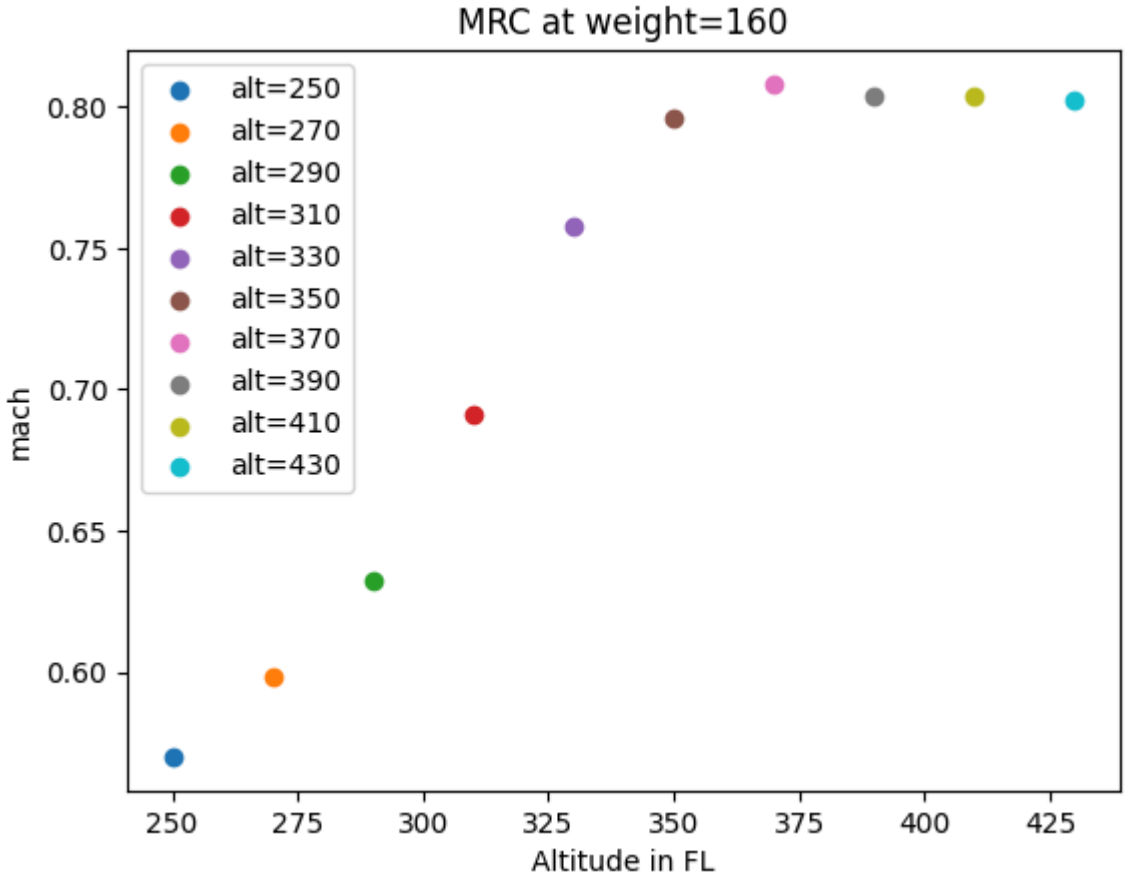
Diagnostic plot to see how linear $\mathrm{MRC}_{\mathrm{mach}}$ is for all altitudes at a given weight.

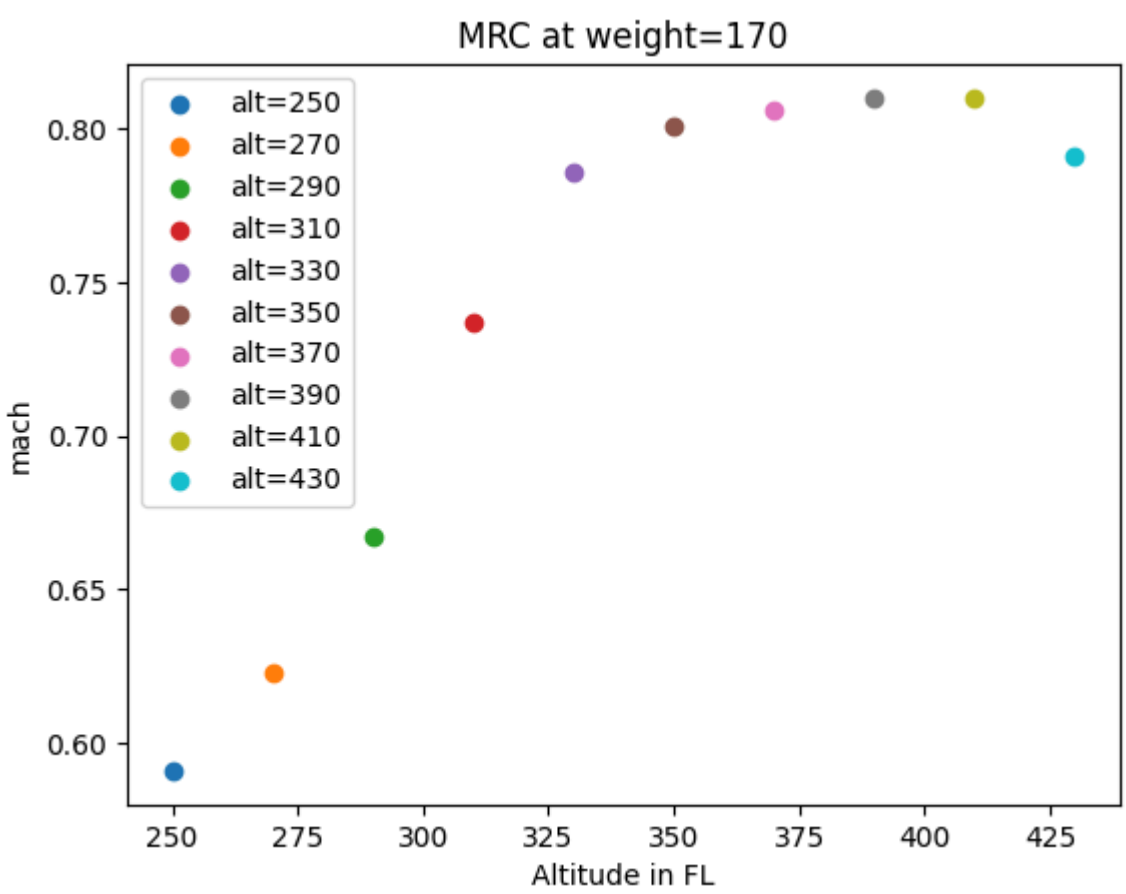
In [ ]:

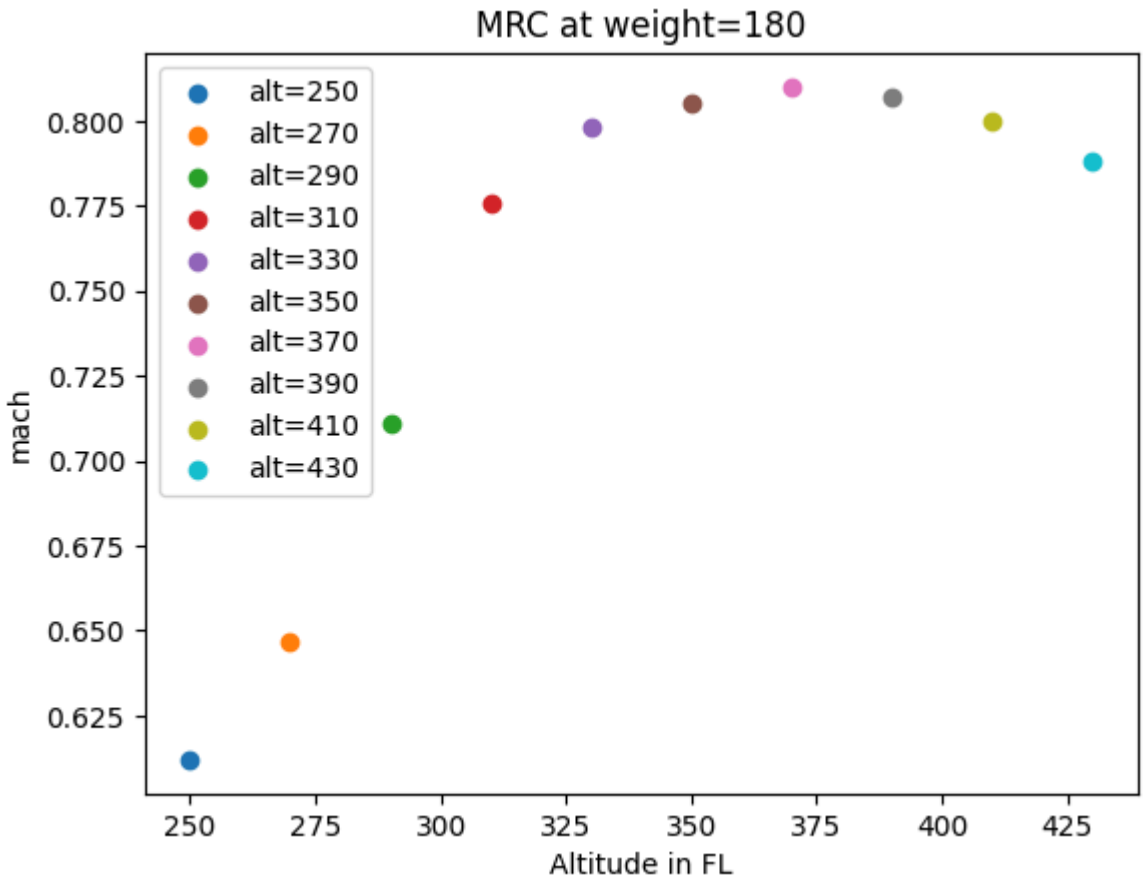
```
if not skip_plots:
  x = np.linspace(250, 430, 10)

  for weight in weights_index:
    plt.title(f"MRC at weight={index2weight(weight)}")
    plt.xlabel("Altitude in FL")
    plt.ylabel("mach")
    for alt in alts_index:
      try:
        plt.scatter(x[alt], curves[alt][weight][0], label=f"alt={index2alt(alt)}")
      except IndexError:
        pass
    plt.legend()
    plt.show()
```

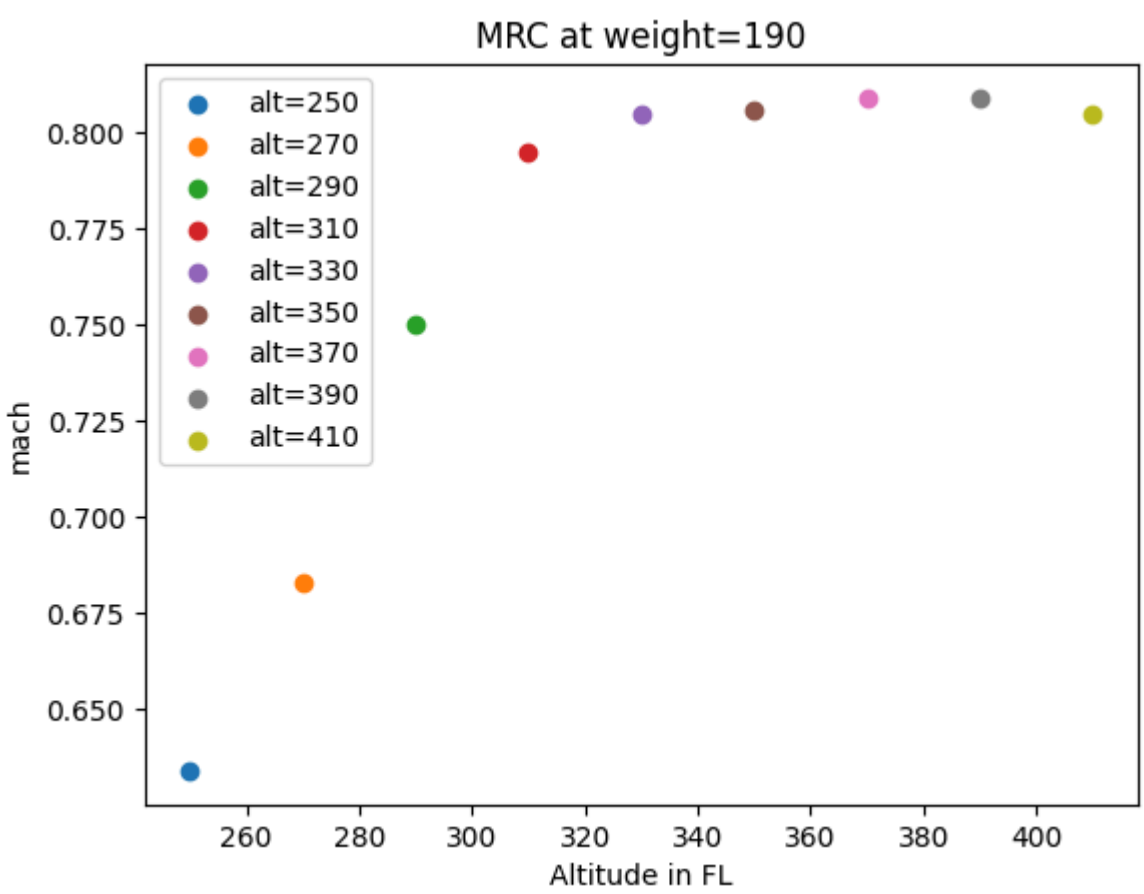


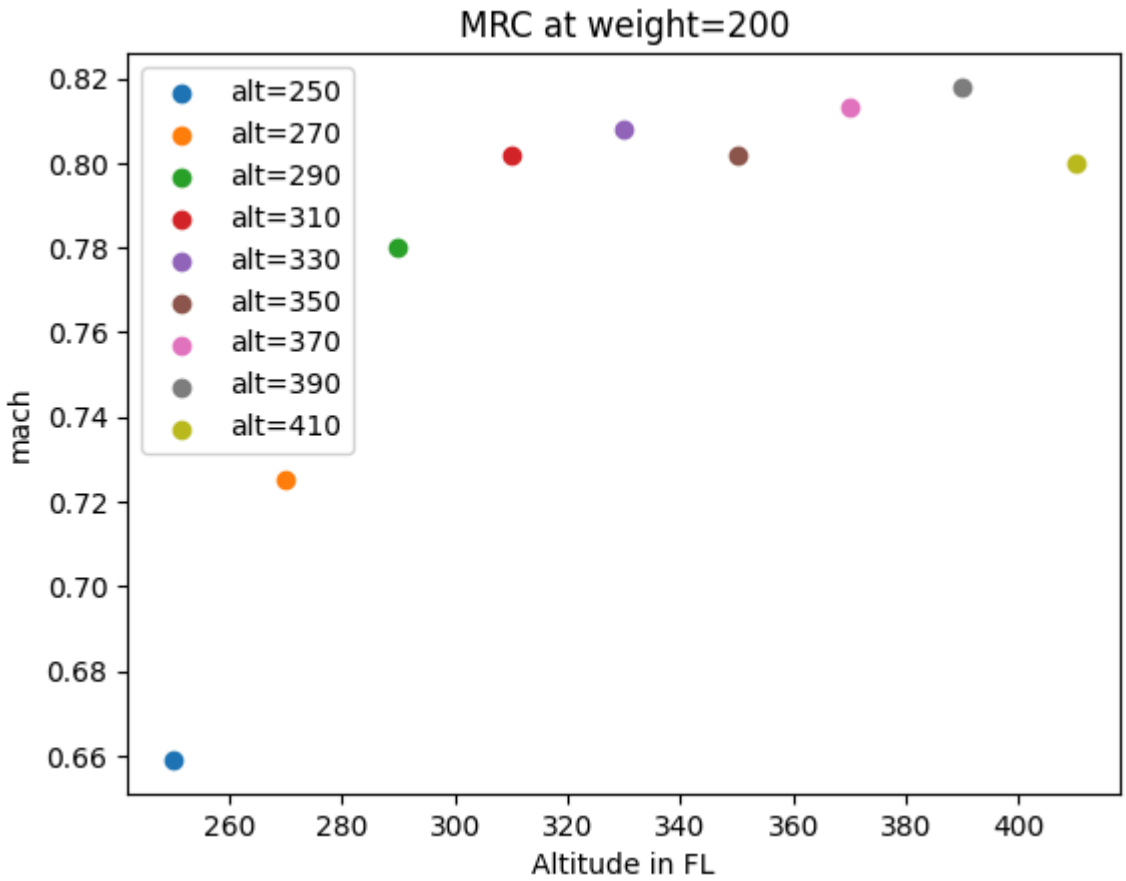

MRC at weight=150
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410
alt=430
mach
0.80
0.75
0.70
0.65
0.60
0.55
250
275
300
325
350
375
400
425
Altitude in FL

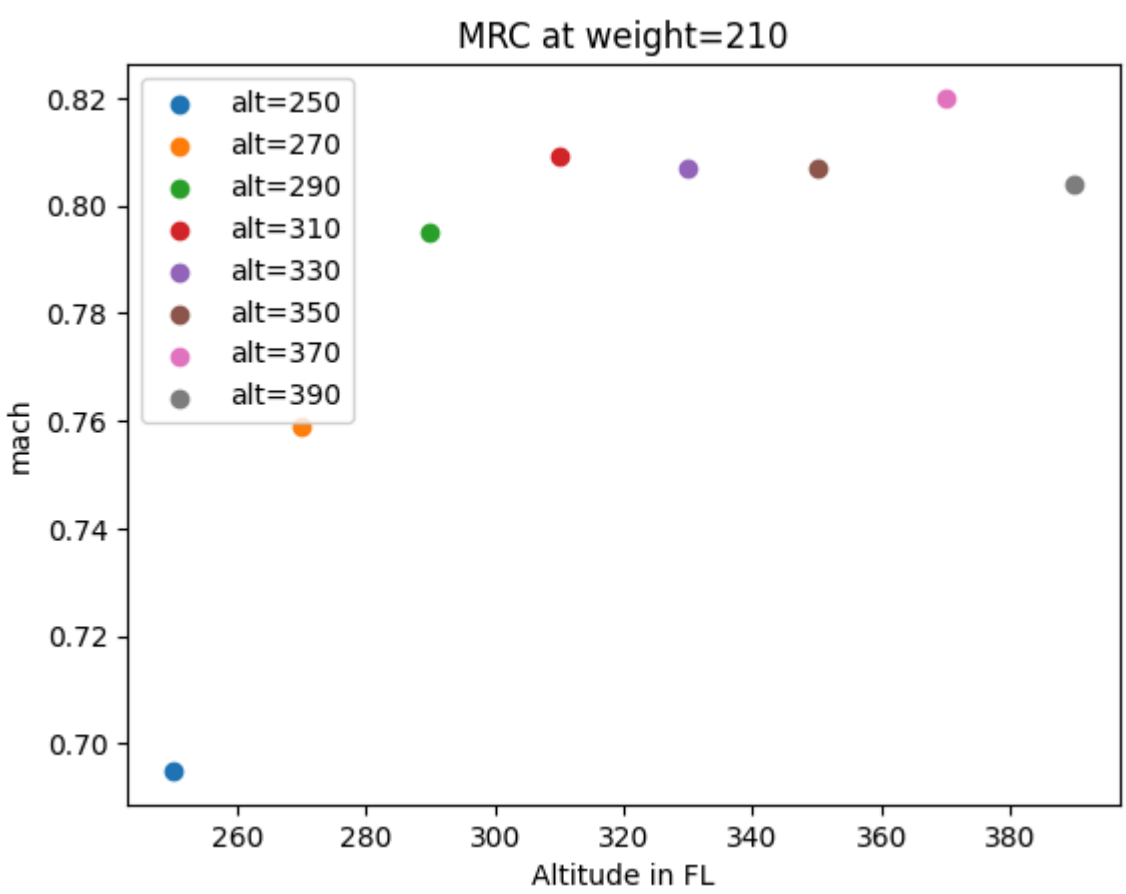

MRC at weight=160
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410
alt=430
mach
0.80
0.75
0.70
0.65
0.60
250
275
300
325
350
375
400
425
Altitude in FL

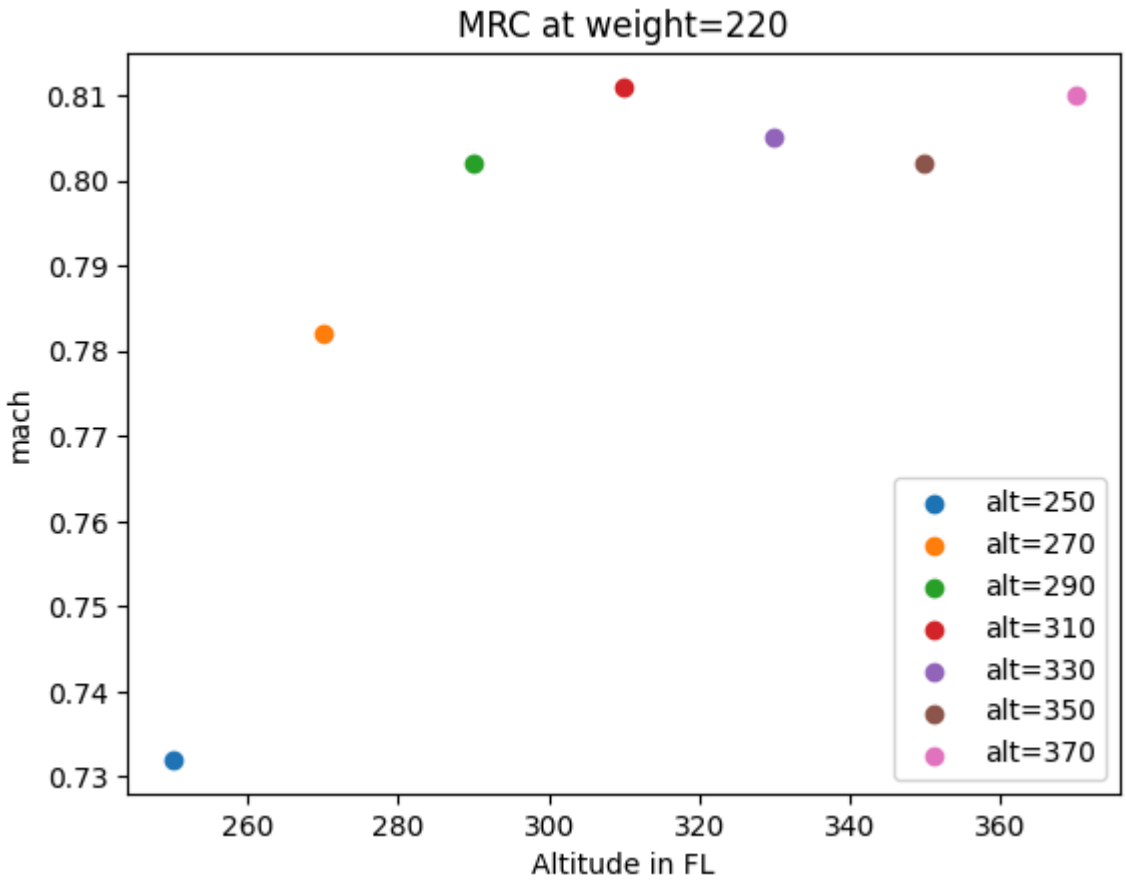

MRC at weight=170
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410
alt=430
0.80
0.75
0.70
0.65
0.60
mach
250
275
300
325
350
375
400
425
Altitude in FL

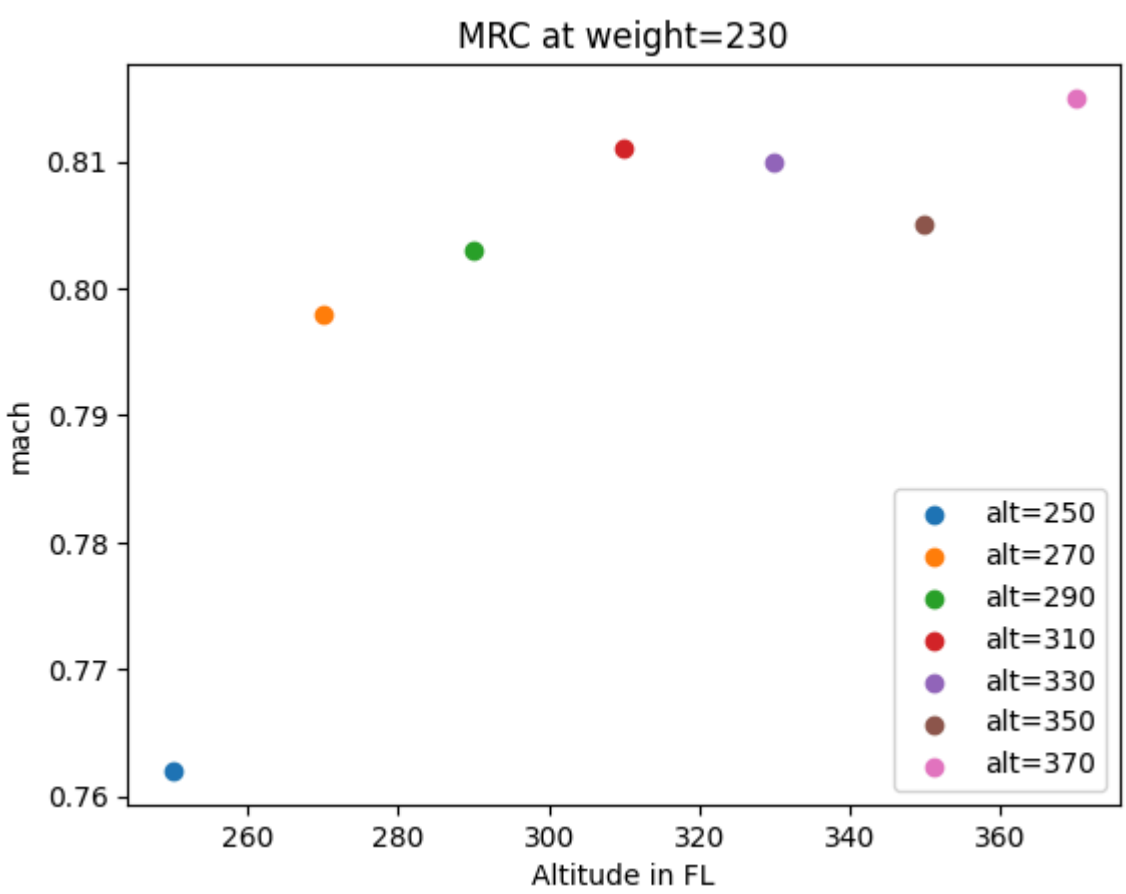

MRC at weight=180
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410
alt=430
0.800
0.775
0.750
0.725
0.700
0.675
0.650
0.625
mach
250
275
300
325
350
375
400
425
Altitude in FL

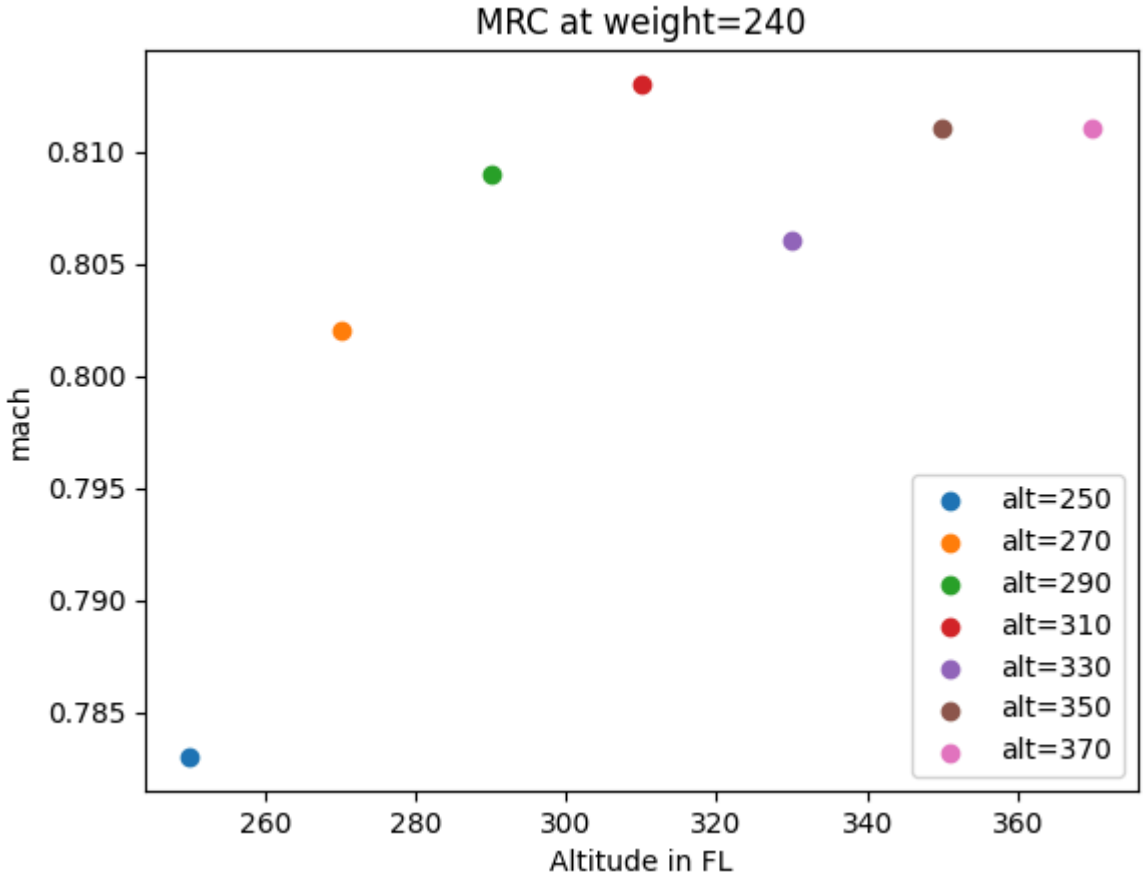

MRC at weight=190
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410
mach
0.800
0.775
0.750
0.725
0.700
0.675
0.650
260
280
300
320
340
360
380
400
Altitude in FL

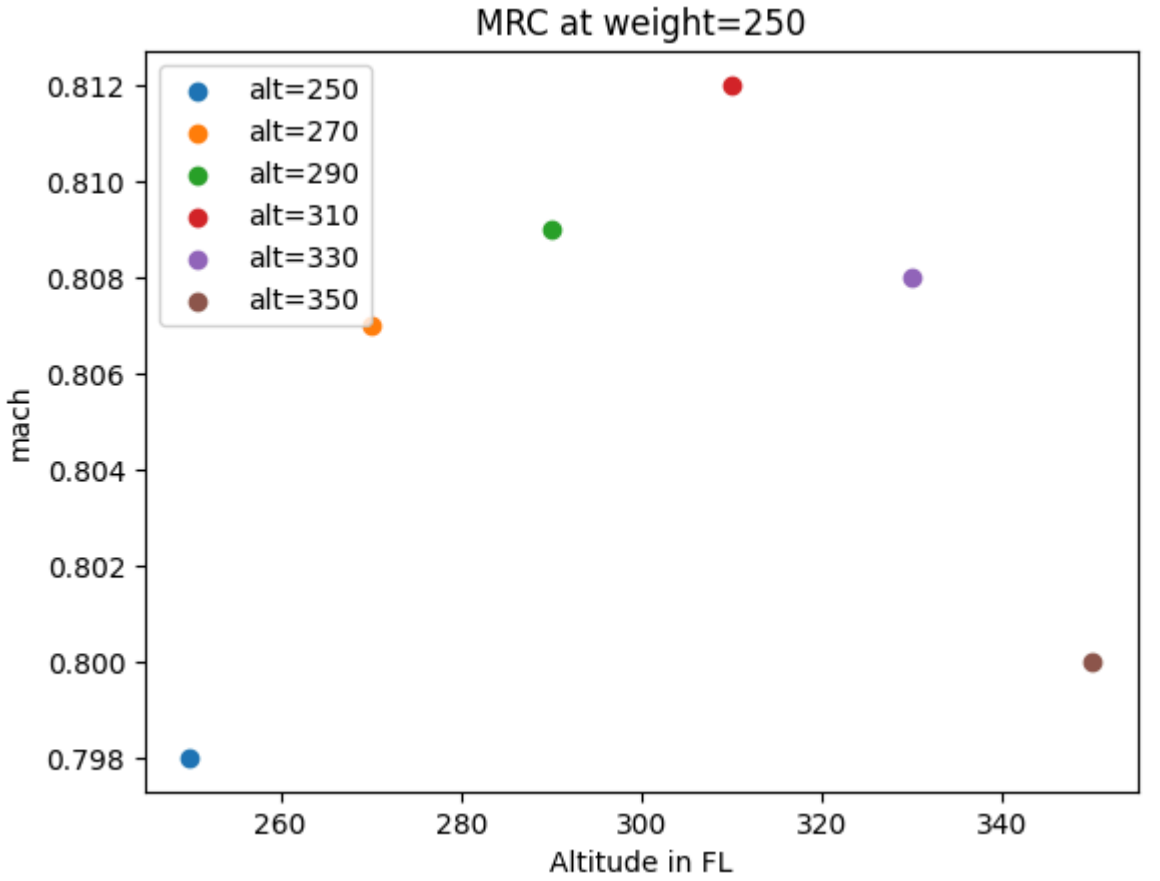

MRC at weight=200
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
alt=410
mach
0.82
0.80
0.78
0.76
0.74
0.72
0.70
0.68
0.66
260
280
300
320
340
360
380
400
Altitude in FL

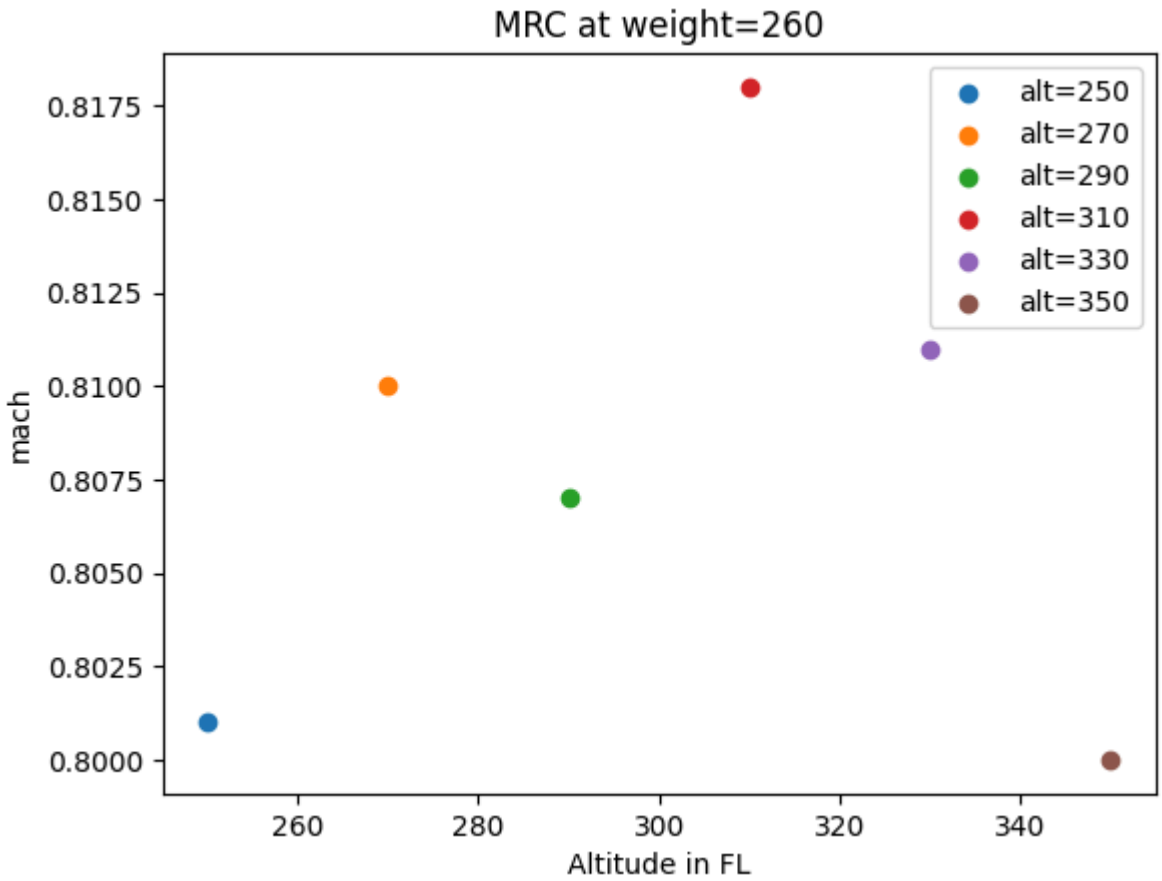

MRC at weight=210
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
alt=390
0.82
0.80
0.78
0.76
0.74
0.72
0.70
mach
260
280
300
320
340
360
380
Altitude in FL

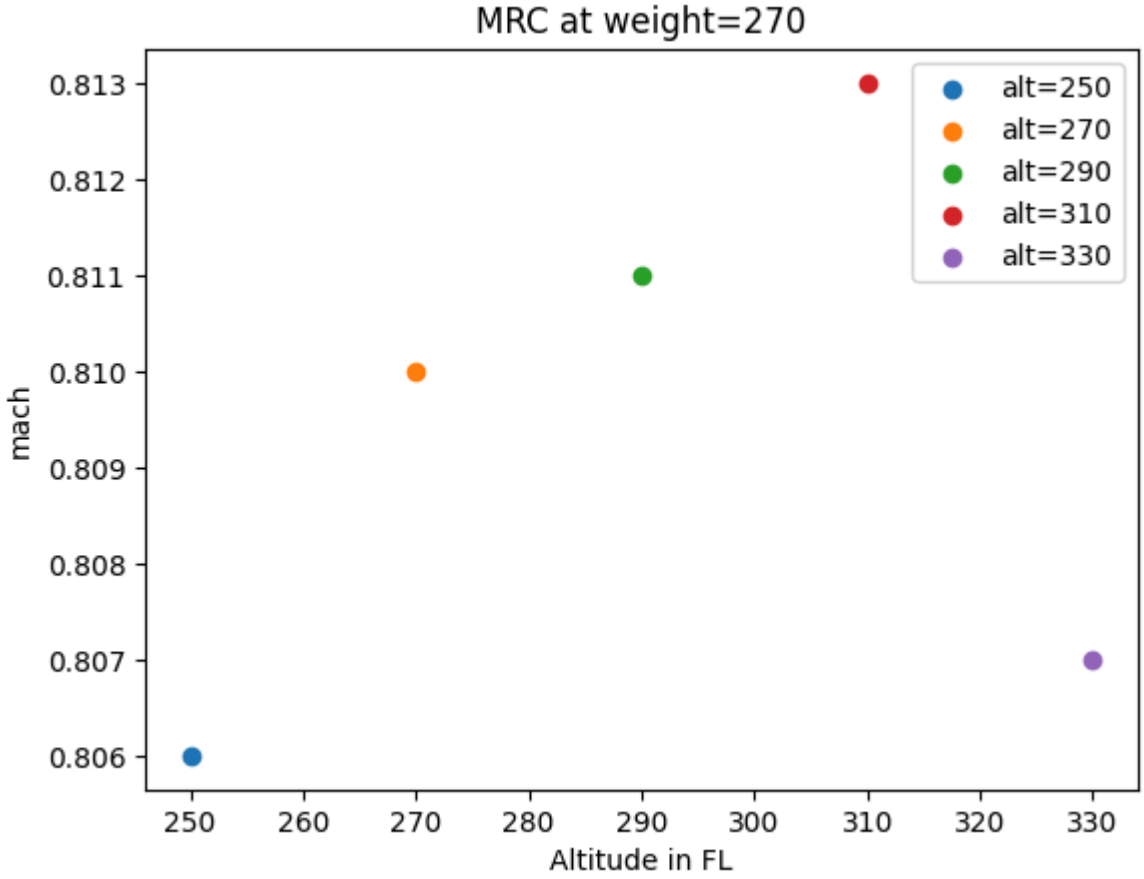

MRC at weight=220
0.81
0.80
0.79
0.78
0.77
0.76
0.75
0.74
0.73
mach
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370
260
280
300
320
340
360
Altitude in FL

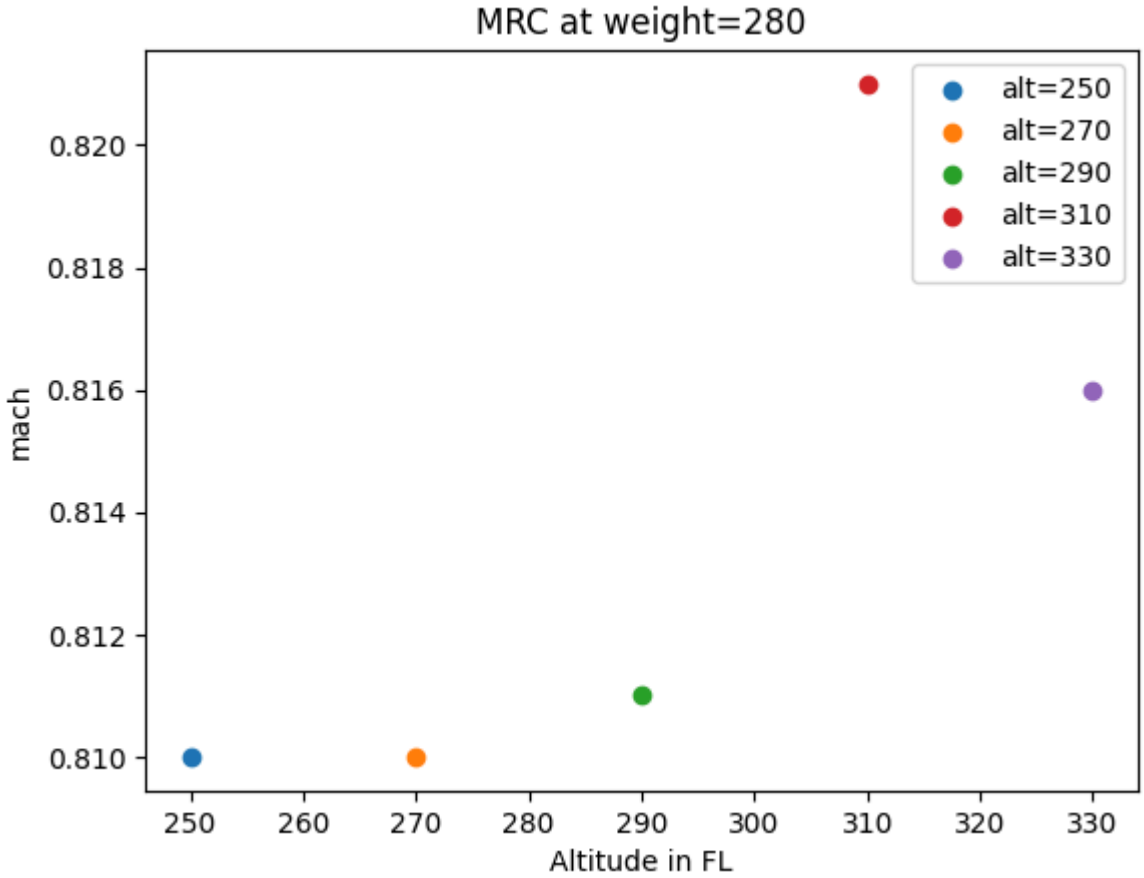

MRC at weight=230
mach
0.81
0.80
0.79
0.78
0.77
0.76
260
280
300
320
340
360
Altitude in FL
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370

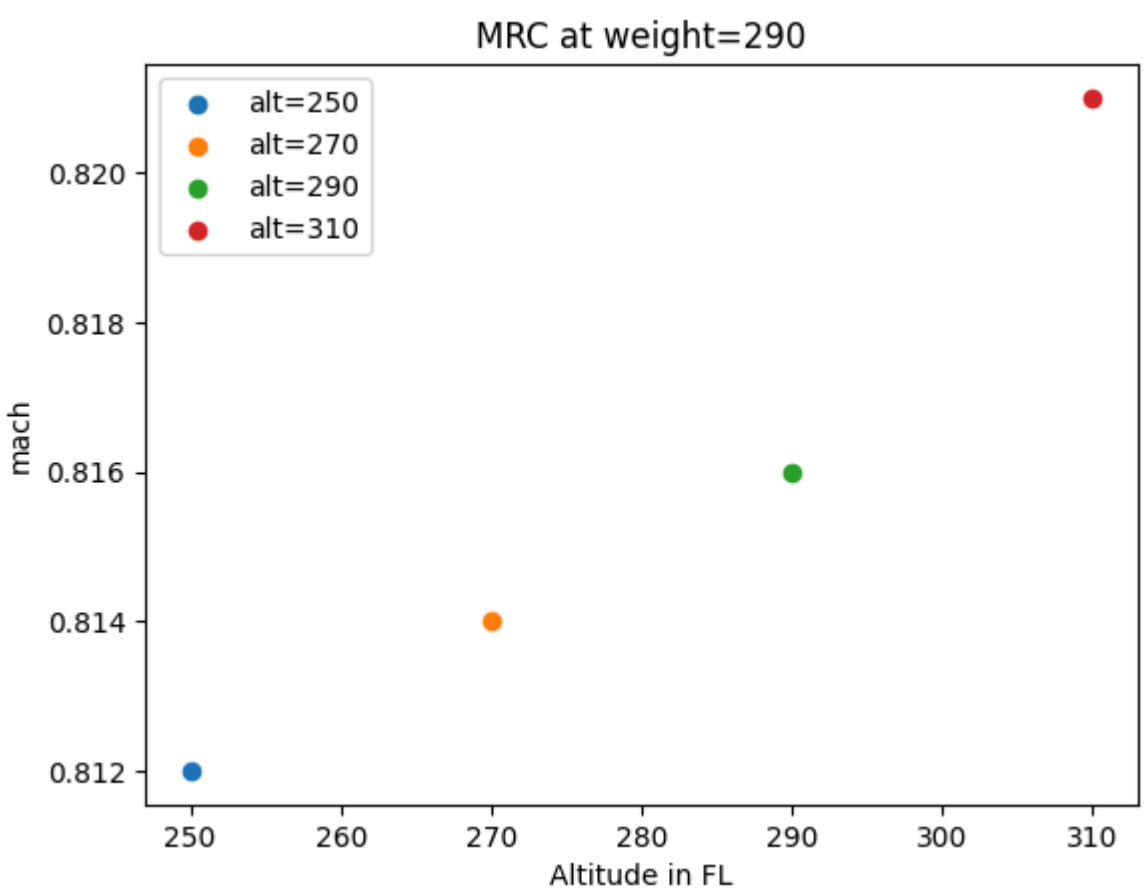

MRC at weight=240
mach
0.810
0.805
0.800
0.795
0.790
0.785
260
280
300
320
340
360
Altitude in FL
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
alt=370


MRC at weight=250
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
0.812
0.810
0.808
0.806
0.804
0.802
0.800
0.798
mach
260
280
300
320
340
Altitude in FL


MRC at weight=260
alt=250
alt=270
alt=290
alt=310
alt=330
alt=350
0.8175
0.8150
0.8125
0.8100
0.8075
0.8050
0.8025
0.8000
mach
260
280
300
320
340
Altitude in FL


MRC at weight=270
alt=250
alt=270
alt=290
alt=310
alt=330
0.813
0.812
0.811
0.810
0.809
0.808
0.807
0.806
mach
250
260
270
280
290
300
310
320
330
Altitude in FL


MRC at weight=280
alt=250
alt=270
alt=290
alt=310
alt=330
0.820
0.818
0.816
0.814
0.812
0.810
mach
250
260
270
280
290
300
310
320
330
Altitude in FL



Output to files.

See `Format description.md` for a description of the file format.

In [ ]:
```
for alt in alts_index:
  file = open(f"ci2mach_{index2alt(alt)}.data", "w")
  for weight in range(16):
    try:
      file.writelines(",".join(str(round(x, 3)) for x in curves[alt][weight].tolist()) + "\n"
    except IndexError:
      pass
```

Demo area.

In [ ]:
```
alt = 290
weight = 250
ci = 80

print(f"{alt=}, {weight=}, {ci=}: {curves[alt2index(alt)][weight2index(weight)][ci]}")
```

```
alt=290, weight=250, ci=80: 0.82278133164632
```