carbon_model.rb

```ruby
require 'leap'
require 'timeframe'
require 'date'
require 'weighted_average'
require 'builder'
require 'flight/carbon_model/fuel_use_equation'
```

Flight carbon model

This model is used by Brighter Planet's carbon emission web service to estimate the **greenhouse gas emissions of passenger air travel**.

Timeframe and date

The model estimates the emissions that occur during a particular `timeframe`. To do this it needs to know the `date` on which the flight occurred. For example, if the `timeframe` is January 2010, a flight that occurred on January 5, 2010 will have emissions but a flight that occurred on February 1, 2010 will not.

Calculations

The final estimate is the result of the **calculations** detailed below. These calculations are performed in reverse order, starting with the last calculation listed and finishing with the `emission` calculation. Each calculation is named according to the value it returns.

Methods

To accomodate varying client input, each calculation may have one or more **methods**. These are listed under each calculation in order from most to least preferred. Each method is named according to the values it requires. If any of these values is not available the method will be ignored. If all the methods for a calculation are ignored, the calculation will not return a value. "Default" methods do not require any values, and so a calculation with a default method will always return a value.

```ruby
module BrighterPlanet
  module Flight
    module CarbonModel
      def self.included(base)
        base.decide :emission, :with => :characteristics do
```

Standard compliance

Each method lists any established calculation standards with which it **complies**. When compliance with a standard is requested, all methods that do not comply with that standard are ignored. This means that any values a particular method requires will have been calculated using a compliant method, because those are the only methods available. If any value did not have a compliant method in its calculation then it would be undefined, and the current method would have been ignored.

Collaboration

Contributions to this carbon model are actively encouraged and warmly welcomed. This library includes a comprehensive test suite to ensure that your changes do not cause regressions. All changes should include test coverage for new functionality. Please see sniff, our emitter testing framework, for more information.

Emission calculation

Returns the `emission` estimate in $kg\ CO_2e$. This is the passenger's share of the total flight emissions that occurred during the `timeframe`.

```
      committee :emission do
```

Emission from fuel use, emission factor, freight share, passengers, multipliers, and date

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
        quorum 'from fuel use, emission factor, freight share, passengers,
multipliers, and date',
          :needs => [:fuel_use, :emission_factor, :freight_share, :passengers,
:seat_class_multiplier, :aviation_multiplier, :date],
          :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics,
timeframe|
            date = characteristics[:date].is_a?(Date) ?
              characteristics[:date] :
              Date.parse(characteristics[:date].to_s)
```

Checks whether the flight occurred during the `timeframe`

```
            if timeframe.include? date
```

Multiplies `fuel use` (kg) by an `emission factor` ($kg\ CO_2e\ /\ kg\ fuel$) and an `aviation multiplier` to give total flight emissions in $kg\ CO_2e$.

```
              characteristics[:fuel_use] * characteristics[:emission_factor] *
characteristics[:aviation_multiplier] *
```

Multiplies by (1 – `freight share`) to take out emissions attributed to freight cargo and mail, leaving emissions attributed to passengers and their baggage

```
                    (1 - characteristics[:freight_share]) /
```

Divides by the number of `passengers` and multiplies by a `seat class multiplier` to give `emission` for the passenger

```
                    characteristics[:passengers] *
characteristics[:seat_class_multiplier]
                  else
```

If the flight did not occur during the `timeframe`, `emission` is zero

```
                    0
                  end
              end
            end
```

Emission factor calculation

Returns the `emission factor` in $kg\ CO_2 / kg\ fuel$.

```
            committee :emission_factor do
```

Emission factor from fuel

```
              quorum 'from fuel',
                :needs => :fuel,
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
                :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
```

Looks up the fuel's `emission factor` ($kg\ CO_2 / l$) and divides by its `density` (kg / l) to give $kg\ CO_2 / kg\ fuel$.

```
                  characteristics[:fuel].co2_emission_factor /
characteristics[:fuel].density
              end
            end
```

Aviation multiplier calculation

Returns the `aviation multiplier`. This approximates the extra climate impact of emissions high in the atmosphere.

```
            committee :aviation_multiplier do
```

Default aviation multiplier

```
              quorum 'default',
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
                :complies => [:ghg_protocol_scope_3, :iso, :tcr] do
```

Uses an `aviation multiplier` of **2.0** after Kollmuss and Crimmins

```
                  2.0
```

(2009).

```
    end
  end
```

Fuel use calculation

Returns the flight's total `fuel use` in kg.

```
  committee :fuel_use do
```

Fuel use from fuel per segment and segments per trip and trips

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Multiplies the `fuel per segment` (kg) by the `segments per trip` and the number of `trips` to give kg.

```
    quorum 'from fuel per segment and segments per trip and trips',
      :needs => [:fuel_per_segment, :segments_per_trip, :trips],

      :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|

        characteristics[:fuel_per_segment] *
characteristics[:segments_per_trip].to_f * characteristics[:trips].to_f
    end
  end
```

Fuel per segment calculation

Returns the `fuel per segment` in kg.

```
  committee :fuel_per_segment do
```

Fuel per segment from adjusted distance per segment and fuel use coefficients

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Uses a third-order polynomial equation to calculate the fuel used per segment:

$(m_3 * d^3) + (m_2 * d^2) + (m_1 * d) +$ endpoint fuel

Where d is the `adjusted distance per segment` and m_3, m_2, m_2, and endpoint fuel are the `fuel use coefficients`.

```
    quorum 'from adjusted distance per segment and fuel use coefficients',
      :needs => [:adjusted_distance_per_segment, :fuel_use_coefficients],

      :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|

        characteristics[:fuel_use_coefficients].m3.to_f *
characteristics[:adjusted_distance_per_segment].to_f ** 3 +
          characteristics[:fuel_use_coefficients].m2.to_f *
characteristics[:adjusted_distance_per_segment].to_f ** 2 +
          characteristics[:fuel_use_coefficients].m1.to_f *
characteristics[:adjusted_distance_per_segment].to_f +
          characteristics[:fuel_use_coefficients].b.to_f
    end
  end
```

Seat class multiplier calculation

```
  committee :seat_class_multiplier do
```

Returns the `seat class multiplier`. This reflects the amount of cabin space occupied by the passenger's seat.

Seat class multiplier from seat class and distance

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Looks up the seat class multiplier based on `distance` and `seat class`.

```
          quorum 'from seat class name and adjusted distance per segment',
            :needs => [:seat_class_name, :adjusted_distance_per_segment],

            :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|

              if characteristics[:adjusted_distance_per_segment] < 244.06

FlightSeatClass.find_by_distance_class_name_and_seat_class_name("Domestic",
"#{characteristics[:seat_class_name]}").multiplier
              elsif characteristics[:adjusted_distance_per_segment] < 863.93

FlightSeatClass.find_by_distance_class_name_and_seat_class_name("Short haul",
"#{characteristics[:seat_class_name]}").multiplier
              else

FlightSeatClass.find_by_distance_class_name_and_seat_class_name("Long haul",
"#{characteristics[:seat_class_name]}").multiplier
              end
          end
```

Seat class multiplier from distance

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Looks up the seat class multiplier based on `distance`.

```
          quorum 'from adjusted distance per segment',
            :needs => :adjusted_distance_per_segment,

            :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|

              if characteristics[:adjusted_distance_per_segment] < 244.06

FlightSeatClass.find_by_distance_class_name_and_seat_class_name("Domestic",
"unknown").multiplier
              elsif characteristics[:adjusted_distance_per_segment] < 863.93

FlightSeatClass.find_by_distance_class_name_and_seat_class_name("Short haul",
"unknown").multiplier
              else

FlightSeatClass.find_by_distance_class_name_and_seat_class_name("Long haul",
"unknown").multiplier
```

```
        end
      end
    end
```

Seat class name calculation

Returns the client-input seat class name.

Adjusted distance per segment calculation

Returns the `adjusted distance per segment` in *nautical miles*.

```
    committee :adjusted_distance_per_segment do
```

Adjusted distance per segment from adjusted distance and segments per trip

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Divides the `adjusted distance` (*nautical miles*) by `segments per trip` to give *nautical miles*.

```
      quorum 'from adjusted distance and segments per trip',
        :needs => [:adjusted_distance, :segments_per_trip],
        :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
        characteristics[:adjusted_distance] /
characteristics[:segments_per_trip]
      end
    end
```

Adjusted distance calculation

Returns the `adjusted distance` in *nautical miles*. The `adjusted distance` accounts for factors that increase the actual distance traveled by real world flights.

```
    committee :adjusted_distance do
```

Adjusted distance from distance, route inefficiency factor, and dogleg factor

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Multiplies `distance` (*nautical miles*) by a `route inefficiency factor` and a `dogleg factor` to give *nautical miles*.

```
      quorum 'from distance, route inefficiency factor, and dogleg factor',
        :needs => [:distance, :route_inefficiency_factor, :dogleg_factor],
        :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
        characteristics[:distance] *
characteristics[:route_inefficiency_factor] * characteristics[:dogleg_factor]
      end
    end
```

Distance calculation

Returns the flight's base `distance` in *nautical miles*.

```
      committee :distance do
```

Distance from airports

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Calculates the great circle distance between the `origin airport` and `destination airport` and converts from *km* to *nautical miles*.

```
        quorum 'from airports',
          :needs => [:origin_airport, :destination_airport],
          :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
            if characteristics[:origin_airport].latitude and
                characteristics[:origin_airport].longitude and
                characteristics[:destination_airport].latitude and
                characteristics[:destination_airport].longitude
characteristics[:origin_airport].distance_to(characteristics[:destination_airport],
:units => :kms).kilometres.to :nautical_miles
            end
        end
```

Distance from distance estimate

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Converts the `distance_estimate` in *km* to *nautical miles*.

```
        quorum 'from distance estimate',
          :needs => :distance_estimate,
          :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
            characteristics[:distance_estimate].kilometres.to :nautical_miles
        end
```

Distance from distance class

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Looks up the distance class' `distance` and converts from *km* to *nautical miles*.

```
        quorum 'from distance class',
          :needs => :distance_class,
          :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
            characteristics[:distance_class].distance.kilometres.to
:nautical_miles
        end
```

Distance from cohort

This should NOT be prioritized over distance estimate or distance class because cohort here never has both airports

Calculates the average `distance` of the `cohort` segments, weighted by their passengers, and converts from *km* to *nautical miles*. Ensure that `distance` > 0

```
      quorum 'from cohort', :needs => :cohort do |characteristics|
        distance = characteristics[:cohort].weighted_average(:distance,
:weighted_by => :passengers).kilometres.to(:nautical_miles)
        distance > 0 ? distance : nil
      end
```

Default distance

Calculates the average `distance` of all segments in the T-100 database, weighted by their passengers, and converts from *km* to *nautical miles*.

```
      quorum 'default' do
        FlightSegment.fallback.distance.kilometres.to :nautical_miles
      end
    end
```

Route inefficiency factor calculation

Returns the `route inefficiency factor`. This is a measure of how much farther real world flights travel than the great circle distance between their origin and destination. It accounts for factors like flight path routing around controlled airspace and circling while waiting for clearance to land.

```
    committee :route_inefficiency_factor do
```

Route inefficiency factor from country

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Looks up the `route inefficiency factor` for the country in which the flight occurs. FIXME: why do we always end up in this quorum even when country is nil?

```
      quorum 'from country',
        :needs => :country,
        :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
          if characteristics[:country].present?
            characteristics[:country].flight_route_inefficiency_factor
          end
      end
```

Default route inefficiency factor

```
      quorum 'default',
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
      :complies => [:ghg_protocol_scope_3, :iso, :tcr] do
```

Uses a `route inefficiency factor` of **10%** based on Kettunen et al. (2005)

```
        Country.fallback.flight_route_inefficiency_factor
    end
  end
```

Dogleg factor calculation

```
  committee :dogleg_factor do
```

Returns the `dogleg factor`. This is a measure of how far out of the way the average layover is compared to a direct flight.

Dogleg factor from segments per trip

```
    quorum 'from segments per trip',
      :needs => :segments_per_trip,
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
      :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
```

Assumes that each layover increases the total flight distance by **25%**.

```
        1.25 ** (characteristics[:segments_per_trip] - 1)
    end
  end
```

Distance estimate calculation

Returns the client-input `distance estimate` in *km*.

Distance class calculation

Returns the client-input distance class.

Fuel use coefficients calculation

```
  committee :fuel_use_coefficients do
```

Returns the `fuel use coefficients`. These are the coefficients of the third-order polynomial equation that describes aircraft fuel use.

Fuel use coefficients from cohort

```
    quorum 'from cohort',
      :needs => :cohort,
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
      :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
```

Calculates the passenger-weighted average fuel use equation for all the flight segments in the cohort

```
        flight_segments = characteristics[:cohort]
```

Initialize a blank fuel use equation for this flight and set a passengers counter to zero

```
        fue = FuelUseEquation.new(0, 0, 0, 0)
        cumulative_passengers = 0

        fs_aircraft_cache = {}
```

For each flight segment in the cohort…

```
        flight_segments.each do |fs|
```

Since we're pulling each member of a cohort in Ruby, rather than just running statistics on the database server level, we're going to cheat a little more

```
          fs_aircraft = (fs_aircraft_cache[fs.aircraft_description] ||=
fs.aircraft.to_a)

          fuel_use_equations = []
          aircraft_classes = []
```

For each aircraft the flight segment refers to…

```
          fs_aircraft.each do |a|
```

If the aircraft is associated with a valid fuel use equation, add that fuel use equation to an array

```
            if a.fuel_use_equation &&
a.fuel_use_equation.valid_fuel_use_equation?
              fuel_use_equations.push(a.fuel_use_equation)
```

Otherwise, if the aircraft's class contains a valid fuel use equation, add the aircraft class to an array

```
            elsif a.aircraft_class &&
a.aircraft_class.valid_fuel_use_equation?
              aircraft_classes.push(a.aircraft_class)
            end
          end
```

Combine the valid fuel use equations and aircraft classes to get an array of equation objects

```
          equation_objects = fuel_use_equations + aircraft_classes
```

If we found at least one valid fuel use equation…

```
          unless equation_objects.empty?
```

Average each coefficient across all the valid fuel use equations, multiply that

```
            fue.m3 += (equation_objects.sum(&:m3) /
```

average by the flight segment's passengers, and add the resulting value to the overall flight fuel use equation

```
equation_objects.length) * fs.passengers
                    fue.m2 += (equation_objects.sum(&:m2) /
equation_objects.length) * fs.passengers
                    fue.m1 += (equation_objects.sum(&:m1) /
equation_objects.length) * fs.passengers
                    fue.b  += (equation_objects.sum(&:b) / equation_objects.length)
* fs.passengers
```

Add the flight segment's passengers to our passengers counter

```
                    cumulative_passengers += fs.passengers
                  end
                end
```

We don't need this cache any more, so we'll help the GC by clearing it

```
                fs_aircraft_cache.clear
```

Check to make sure at least one of the segments had passengers and a valid fuel use equation

```
                if cumulative_passengers > 0
```

Divide each coefficient in our overall fuel use equation by the passengers counter and return the result

```
                  fue.m3 /= cumulative_passengers
                  fue.m2 /= cumulative_passengers
                  fue.m1 /= cumulative_passengers
                  fue.b /= cumulative_passengers
                  fue
                end
            end
```

Fuel use coefficients from aircraft

```
            quorum 'from aircraft',
              :needs => :aircraft,
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
              :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
```

Looks up the aircraft's `fuel use coefficients`.

```
                if equation = characteristics[:aircraft].fuel_use_equation
                  fuel_use = equation.valid_fuel_use_equation? ?
FuelUseEquation.new(equation.m3, equation.m2, equation.m1, equation.b) : nil
                  fuel_use
                end
            end
```

Fuel use coefficients from aircraft class

```
          quorum 'from aircraft class',
            :needs => :aircraft_class,
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
            :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
```

Looks up the aircraft class's `fuel use coefficients`.

```
              ac = characteristics[:aircraft_class]
              fuel_use = ac.valid_fuel_use_equation? ? FuelUseEquation.new(ac.m3,
ac.m2, ac.m1, ac.b) : nil
              fuel_use
          end
```

Default fuel use coefficients

```
          quorum 'default',
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
            :complies => [:ghg_protocol_scope_3, :iso, :tcr] do
```

Calculates the average `fuel use coefficients` of the aircraft used by all segments in the T-100 database, weighted by the segment passengers.

```
              FuelUseEquation.new AircraftFuelUseEquation.fallback.m3,
AircraftFuelUseEquation.fallback.m2, AircraftFuelUseEquation.fallback.m1,
AircraftFuelUseEquation.fallback.b
          end
        end
```

Fuel calculation

```
        committee :fuel do
```

Returns the `fuel`.

Fuel from client input

Complies: All

Uses the client-input fuel.

Default fuel

```
          quorum 'default',
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
            :complies => [:ghg_protocol_scope_3, :iso, :tcr] do
```

Assumes the flight uses **Jet Fuel**.

```
              Fuel.find_by_name 'Jet Fuel'
          end
          end
```

Passengers calculation

Returns the number of `passengers`.

```
          committee :passengers do
```

Passengers from seats and load factor

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Multiplies the number of `seats` by the `load factor`.

```
            quorum 'from seats and load factor',
              :needs => [:seats, :load_factor],

              :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|

                (characteristics[:seats] * characteristics[:load_factor]).round
            end
          end
```

Seats calculation

Returns the number of `seats`.

```
          committee :seats do
```

Seats from seats estimate

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Uses the client-input estimate of the number of `seats`.

```
            quorum 'from seats estimate',
              :needs => :seats_estimate,

              :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|

                characteristics[:seats_estimate] > 0 ?
characteristics[:seats_estimate] : nil
            end
```

Seats from cohort

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Calculates the average number of `seats` of the `cohort` segments, weighted by their passengers. Ensure that `seats` > 0

```
            quorum 'from cohort',
              :needs => :cohort,

              :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|

                seats =
characteristics[:cohort].weighted_average(:seats_per_flight, :weighted_by =>
:passengers)
```

```
        seats > 0 ? seats : nil
    end
```

Seats from aircraft

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Looks up the aircraft's average number of `seats`.

```
    quorum 'from aircraft',
      :needs => :aircraft,
      :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
        seats = characteristics[:aircraft].seats
        seats.present? && seats > 0 ? seats : nil
    end
```

Seats from aircraft class

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Looks up the aircraft class's average number of `seats`.

```
    quorum 'from aircraft class',
      :needs => :aircraft_class,
      :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
        seats = characteristics[:aircraft_class].seats
        seats.present? && seats > 0 ? seats : nil
    end
```

Default seats

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Calculates the average number of `seats` of all segments in the T-100 database, weighted by their passengers.

```
    quorum 'default',
      :complies => [:ghg_protocol_scope_3, :iso, :tcr] do
        FlightSegment.fallback.seats_per_flight
    end
  end
```

Aircraft Class calculation

Returns the aircraft class.

```
  committee :aircraft_class do
```

Aircraft class from aircraft

```
    quorum 'from aircraft',
      :needs => :aircraft,
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
        :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
```

Looks up the aircraft's aircraft_class.

```
          characteristics[:aircraft].aircraft_class
      end
    end
```

Seats estimate calculation

Returns the client-input `seats estimate`.

Load factor calculation

Returns the `load factor`. This is the portion of available seats that are occupied.

```
    committee :load_factor do
```

Load factor from client input

Complies: All

Uses the client-input `load factor`.

Load factor from cohort

```
      quorum 'from cohort',
        :needs => :cohort,
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
        :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
```

Calculates the average `load factor` of the `cohort` segments, weighted by their passengers. Ensure that `load_factor` > 0

```
          load_factor =
characteristics[:cohort].weighted_average(:load_factor, :weighted_by =>
:passengers)
          load_factor > 0 ? load_factor : nil
      end
```

Default load factor

```
      quorum 'default',
```

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
        :complies => [:ghg_protocol_scope_3, :iso, :tcr] do
```

Calculates the average `load factor` of all segments in the T-100 database, weighted by their passengers.

```
          FlightSegment.fallback.load_factor
        end
      end
```

Freight share calculation

Returns the `freight share`. This is the percent of the total aircraft weight that is freight cargo and mail (as opposed to passengers and their baggage).

```
      committee :freight_share do
```

Freight share from cohort

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Calculates the average `freight share` of the `cohort` segments, weighted by their passengers. Don’t need checks because zero is a valid `freight share`

```
        quorum 'from cohort',
          :needs => :cohort,
          :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
            characteristics[:cohort].weighted_average(:freight_share,
:weighted_by => :passengers)
        end
```

Default freight share

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Calculates the average `freight share` of all segments in the T-100 database, weighted by their passengers.

```
        quorum 'default',
          :complies => [:ghg_protocol_scope_3, :iso, :tcr] do
            FlightSegment.fallback.freight_share
        end
      end
```

Trips calculation

Returns the number of `trips`. A one-way flight has one trip; a round-trip flight has two trips.

```
      committee :trips do
```

Trips from client input

Complies: All

Uses the client-input number of `trips`.

Default trips

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Uses an average number of `trips` of **1.7** calculated from the BTS Origin and Destination Survey.

```
      quorum 'default',
       :complies => [:ghg_protocol_scope_3, :iso, :tcr] do


        1.7
      end
     end
```

Country calculation

Returns the country in which a flight occurs.

```
     committee :country do
```

Country from origin airport and destination airport

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Checks whether the flight's `origin airport` and `destination airport` are within the same country. If so, that country is the `country`.

```
      quorum 'from origin airport and destination airport',
       :needs => [:origin_airport, :destination_airport],


       :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|


        if characteristics[:origin_airport].country ==
characteristics[:destination_airport].country
         characteristics[:origin_airport].country
        end
      end
     end
```

Cohort calculation

Returns the `cohort`. This is a set of flight segment records in the T-100 database that match certain client-input values.

```
     committee :cohort do
      quorum 'from row_hash', :needs => [:flight_segment_row_hash] do
|characteristics|
       FlightSegment.where(:row_hash =>
characteristics[:flight_segment_row_hash].value).to_cohort
      end
```

Cohort from segments per trip and input

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

```
      quorum 'from segments per trip and input',
       :needs => :segments_per_trip, :appreciates => [:origin_airport,
:destination_airport, :aircraft, :airline, :date],


       :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics|
```

Only assemble a cohort if the flight is direct

```
                if characteristics[:segments_per_trip] == 1
                  cohort = {}
                  provided_characteristics = []
                  date = characteristics[:date].is_a?(Date) ?
characteristics[:date] : Date.parse(characteristics[:date].to_s)
```

We'll want to restrict the cohort to flight segments that occurred the same year as the flight or the previous year. We need to include the previous year because our flight segment data lags by about 6 months.

```
                  relevant_years = [date.year - 1, date.year]
```

If we have both an origin and destination airport…

```
                  if characteristics[:origin_airport].present? and
characteristics[:destination_airport].present?
```

If either airport is in the US, use airport iata code to assemble a cohort of BTS flight segments

```
                    if characteristics[:origin_airport].country_iso_3166_code ==
"US" or characteristics[:destination_airport].country_iso_3166_code == "US"
```

NOTE: It's possible that the origin/destination pair won't appear in our database and we'll end up using a cohort based just on origin. If that happens, even if the origin is not in the US we still don't want to use origin airport city, because we know the flight was going to the US and ICAO segments never touch the US.

```
                      provided_characteristics.push [:origin_airport_iata_code,
characteristics[:origin_airport].iata_code]
                      provided_characteristics.push
[:destination_airport_iata_code, characteristics[:destination_airport].iata_code]
```

If neither airport is in the US, use airport city to assemble a cohort of ICAO flight segments FIXME TODO: deal with cities in multiple countries that share a name pushing country works if we're trying to go from Mexico City to Barcelona, Spain and so the cohort should NOT include flights to Barcelona, Venezuela BUT it won't work if we're trying to go from Montreal to London, Canada – there are no direct flights to London, Canada but there ARE flights to London, United Kingdom so we end up with those

```
                    else
                      provided_characteristics.push [:origin_airport_city,
characteristics[:origin_airport].city]
                      provided_characteristics.push [:origin_country_iso_3166_code,
characteristics[:origin_airport].country_iso_3166_code]
                      provided_characteristics.push [:destination_airport_city,
characteristics[:destination_airport].city]
                      provided_characteristics.push
[:destination_country_iso_3166_code,
characteristics[:destination_airport].country_iso_3166_code]
                    end
```

Also use aircraft description and airline name

```
                    if characteristics[:aircraft].present?
                      provided_characteristics.push [:aircraft_description,
characteristics[:aircraft].flight_segments_foreign_keys]
                    end
                    if characteristics[:airline].present?
                      provided_characteristics.push [:airline_name,
characteristics[:airline].name]
                    end
```

To assemble a cohort, we start with all the flight segments that are the same year as the flight or the previous year. Then we find all the segments that match the input `origin_airport`, `destination_airport`, `aircraft`, and `airline`. If no segments match all the inputs, we drop the last input (initially `airline`) and try again. We continue until some segments match or no inputs remain.

```
                    cohort = FlightSegment.where(:year =>
relevant_years).strict_cohort(*provided_characteristics)
```

Ignore the cohort if none of its flight segments have any passengers TODO: make ‘passengers > 0’ a constraint once cohort_scope supports non-hash constraints

```
                    if cohort.any? && cohort.any? { |fs| fs.passengers.nonzero? }
                      cohort
                    else
                      nil
                    end
```

If we have either origin or destination but not both… NOTE: This needs to be a special case because if we had neither origin nor destination, generated separate BTS and ICAO cohorts, and combined them the resulting cohort would have two copies of each flight segment.

```
                  elsif characteristics[:origin_airport].present? or
characteristics[:destination_airport].present?
```

First use airport iata code to assemble a cohort of BTS flight segments

```
                    if characteristics[:origin_airport].present?
                      provided_characteristics.push [:origin_airport_iata_code,
characteristics[:origin_airport].iata_code]
                      provided_characteristics.push [:origin_country_iso_3166_code,
characteristics[:origin_airport].country_iso_3166_code]
                    end

                    if characteristics[:destination_airport].present?
                      provided_characteristics.push
[:destination_airport_iata_code, characteristics[:destination_airport].iata_code]
                      provided_characteristics.push
[:destination_country_iso_3166_code,
characteristics[:destination_airport].country_iso_3166_code]
                    end

                    if characteristics[:aircraft].present?
                      provided_characteristics.push [:aircraft_description,
```

Note: can't use where(:year => relevant_years) here because then when we combine the cohorts you get WHERE year IN (*relevant_years*) OR *other conditions* which returns every flight segment where(:year => relevant_years)

Then use airport city to assemble a cohort of ICAO flight segments FIXME TODO: deal with cities in multiple countries that share a name pushing country works if we're trying to go from Mexico City to Barcelona, Spain and so the cohort should NOT include flights to Barcelona, Venezuela BUT it won't work if we're trying to go from Montreal to London, Canada – there are no direct flights to London, Canada but there ARE flights to London, United Kingdom so we end up with those

```
characteristics[:aircraft].flight_segments_foreign_keys]
                    end

                    if characteristics[:airline].present?
                      provided_characteristics.push [:airline_name,
characteristics[:airline].name]
                    end



                    bts_cohort =
FlightSegment.strict_cohort(*provided_characteristics)




                    provided_characteristics = []
                    if characteristics[:origin_airport].present?
                      provided_characteristics.push [:origin_airport_city,
characteristics[:origin_airport].city]
                      provided_characteristics.push [:origin_country_iso_3166_code,
characteristics[:origin_airport].country_iso_3166_code]
                    end

                    if characteristics[:destination_airport].present?
                      provided_characteristics.push [:destination_airport_city,
characteristics[:destination_airport].city]
                      provided_characteristics.push
[:destination_country_iso_3166_code,
characteristics[:destination_airport].country_iso_3166_code]
                    end

                    if characteristics[:aircraft].present?
                      provided_characteristics.push [:aircraft_description,
characteristics[:aircraft].flight_segments_foreign_keys]
                    end

                    if characteristics[:airline].present?
                      provided_characteristics.push [:airline_name,
characteristics[:airline].name]
                    end

                    icao_cohort =
FlightSegment.strict_cohort(*provided_characteristics)
```

Combine the two cohorts, making sure to restrict to relevant years Note: cohort_scope 0.2.1 provides cohort + cohort => cohort; cohort.where() => relation; relation.to_cohort => cohort

```
                    cohort = (bts_cohort + icao_cohort).where(:year =>
relevant_years).to_cohort
```

Ignore the resulting cohort if none of its flight segments have any passengers TODO: make 'passengers > 0' a constraint once cohort_scope supports non-hash constraints

```
                    if cohort.any? && cohort.any? { |fs| fs.passengers.nonzero? }
                      cohort
                    else
                      nil
                    end
```

If we have neither origin nor destination…

```
                  else
```

Use aircraft description and airline name to assemble a cohort

```
                    if characteristics[:aircraft].present?
                      provided_characteristics.push [:aircraft_description,
characteristics[:aircraft].flight_segments_foreign_keys]
                    end

                    if characteristics[:airline].present?
                      provided_characteristics.push [:airline_name,
characteristics[:airline].name]
                    end

                    cohort = FlightSegment.where(:year =>
relevant_years).strict_cohort(*provided_characteristics)
```

Ignore the cohort if none of its flight segments have any passengers TODO: make 'passengers > 0' a constraint once cohort_scope supports non-hash constraints

```
                    if cohort.any? && cohort.any? { |fs| fs.passengers.nonzero? }
                      cohort
                    else
                      nil
                    end
                  end
                end
            end
          end
```

Origin airport calculation

Returns the client-input origin airport.

Destination airport calculation

Returns the client-input destination airport.

Aircraft calculation

Returns the client-input of aircraft.

Airline calculation

Returns the client-input airline operating the flight.

Segments per trip calculation

Returns the `segments per trip`. Direct flights have a single segment per trip. Indirect flights with one or more layovers have two or more segments per trip.

```
committee :segments_per_trip do
```

Segments per trip from client input

Complies: All

Uses the client-input `segments per trip`.

Default segments per trip

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Uses an average `segments per trip` of **1.68**, calculated from the BTS Origin and Destination Survey.

```
  quorum 'default',
    :complies => [:ghg_protocol_scope_3, :iso, :tcr] do
      1.68
  end
end
```

Date calculation

Returns the `date` on which the flight occurred.

```
committee :date do
```

Date from client input

Complies: All

Uses the client-input `date`.

Date from timeframe

Complies: GHG Protocol Scope 3, ISO-14064-1, Climate Registry Protocol

Assumes the flight occurred on the first day of the `timeframe`.

```
            quorum 'from timeframe',
              :complies => [:ghg_protocol_scope_3, :iso, :tcr] do |characteristics, timeframe|
                timeframe.from
            end
          end
```

Timeframe calculation

Returns the `timeframe`. This is the period during which to calculate emissions.

Timeframe from client input

Complies: All

Uses the client-input `timeframe`.

Default timeframe

Complies: All

Uses the current calendar year.

```
        end
      end
    end
  end
end
```