Energy and Carbon Calculator for Homes Technical Overview

NZGBC
TE KAUNIHERA HANGANGA TAUTAIAO

Introduction

At NZGBC, we want New Zealanders to have better performing homes and this means that the design of a house is critical. The Energy and Carbon Calculator for Homes (ECCHO) is a digital online method enabling calculation of the overall heating and cooling demands of a home, together with overall energy (fuel) consumption and carbon emissions based on space heating and hot water systems, lighting, plug loads and refrigerant losses.

The ECCHO modelling methodology is not intended to provide all the answers when designing homes – but this overview provides information about how calculations in ECCHO are performed and aims to help ECCHO users understand the methods and limitations of the ECCHO model. Knowing more about 'what's under the hood' can increase users' understanding of the principles underpinning ECCHO and what factors are important to consider when designing homes. It also means home designers can better understand whether ECCHO is suitable for their project. For example, ECCHO cannot calculate dynamic or zonal effects so it can't be used to solve those types of issues for instance, when a house is at risk of overheating only in the afternoon or only on the western side.

ECCHO can be used in conjunction with Homestar to meet v5 requirements and help create a comfortable, healthier, more energy efficient home.

In creating, ECCHO, NZGBC relied on the same conventions and methods as the Passive House Institute's Passive House Planning Package (PHPP) (version 9.6a), Excel-based energy modelling software. NZGBC created ECCHO to be an advanced digital online methodology (written in Python) and is independent of PHPP. The Python code summaries in the final appendix, will allow developers to better maintain ECCHO into the future.

Results from any ECCHO calculations should be interpreted by professionals who understand how to use the model, and it is recommended that ECCHO is used in conjunction with other home design and construction methodologies. All diagrams in this overview are illustrative only.

For more information about ECCHO, please contact homestar@nzgbc.org.nz

This overview has been co-authored by Sam Archer and Jason Quinn of Sustainable Engineering Ltd. Sam is the Director of Market Transformation at NZGBC and developer of the ECCHO model and Jason is a Passive House Home Designer and PHI Accredited Certifier.

Overall calculation

The overall calculation method in ECCHO pulls in dwelling data from the ECCHO database and climate data. It calculates an energy balance for heating, cooling and overheating; and then the overall energy (fuel) consumption and carbon emissions based on space heating and hot water systems, lighting, plug loads and refrigerant losses.

This overview and the ECCHO code reference TFA (Treated Floor Area) but Homestar uses CFA (Conditioned Floor Area). These are treated as the same value in this text but do note they can differ. In a few cases that were compared, CFA is approximately 5% higher than TFA. This does not affect the overall energy demand but does change the per square metre metrics, as the area used as a reference can be larger.

The core heating and cooling demand calculation used by ECCHO is based on ISO 13790:2008. See Figure 1 below. This standard describes a method for carrying out a monthly heat balance calculation using average monthly climate data for 19 locations in New Zealand. Fundamentally this uses conductive heat loss/gain on a monthly basis with correction factors for the 'useful' gains/losses. These correction factors are based on dynamic models derived similarly to ISO52016. Solar gains on transparent surfaces use monthly averages for vertical surfaces for each compass direction (precalculated) and a horizontal surface to calculate the monthly solar gain for the specific transparent surface orientation and tilt. This is similar to the way ISO52010 pre-calculates monthly solar inputs but allows for any surface orientation and tilt with smooth changes between values. This compares to lumping surfaces into the closest orientation and calculating as ISO52010 suggests.

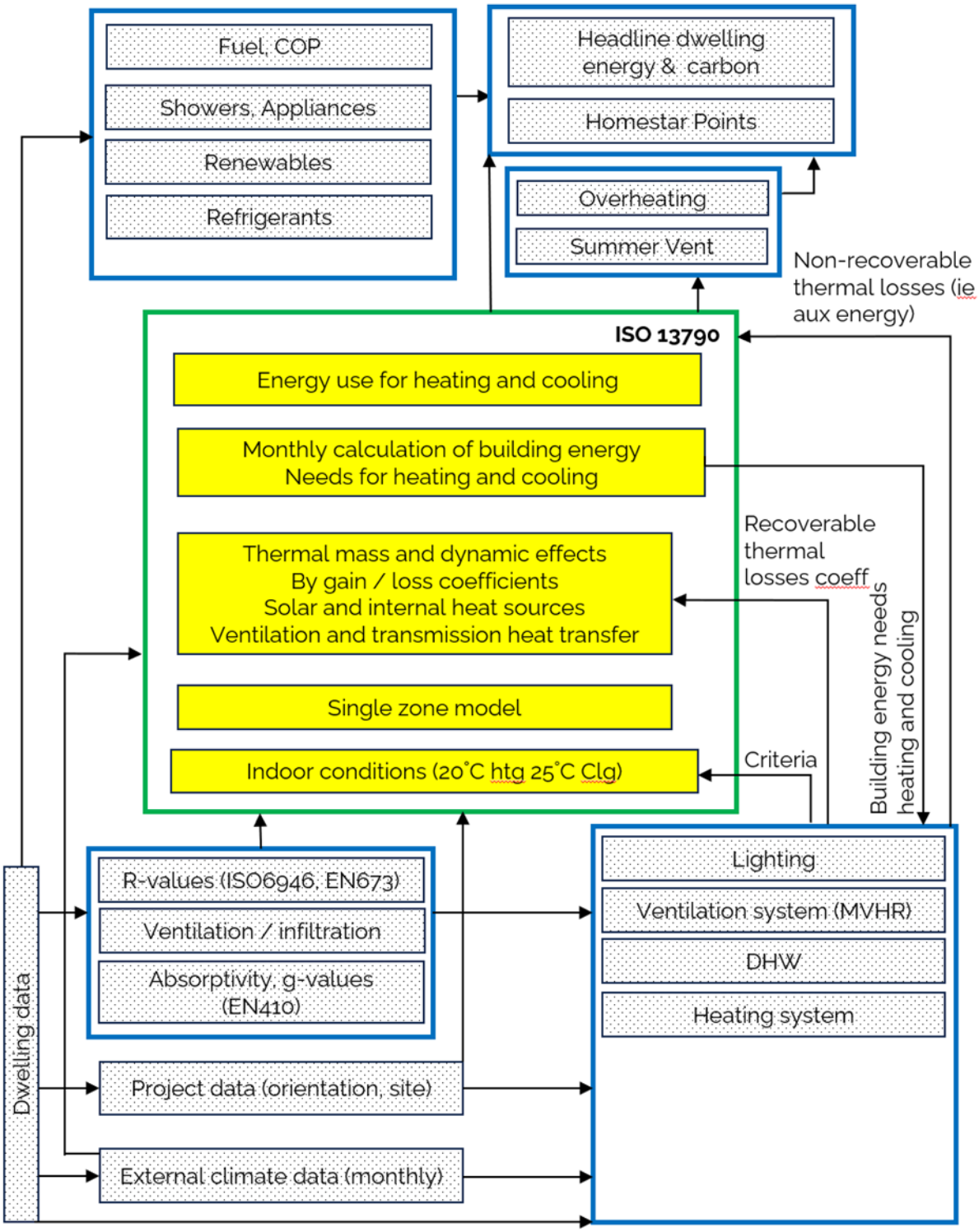

Figure 1. ECCHO online overall flowchart

Space heating calculation – monthly method

Heating Balance

Annual space heating demand (kWh/year) is calculated in ECCHO by estimating and then summing, the residual heat input required to balance the following gains and losses for each month:

- transmission heat transfer between the conditioned space and the external environment, governed by the difference between the temperature of the conditioned zone and the (monthly average) external temperature;
- ventilation heat transfer (by natural ventilation or by a mechanical ventilation system), governed by the difference between the temperature of the conditioned zone and the supply air temperature;
- radiative heat losses to the sky and convection losses from air movement across the external surfaces;
- internal heat gains (including negative gains from heat sinks), for instance from persons, appliances, lighting and heat dissipated in or absorbed by heating, cooling, hot water or ventilation systems (multiplied by the gains utilisation factor);
- solar heat gains (which can be direct, e.g. through windows, or indirect, e.g. via absorption in opaque building elements); and
- storage of heat in, or release of stored heat from, the mass of the building;

This is a quasi-steady state method calculating the heat balance over each month and the dynamic effects are considered by an empirically determined gain utilisation factor. Note this generally follows the monthly method from ISO13790, not the annual method, although the results are presented as annual sums.

As noted, [PHPP 2015 pg. 165] the annual heating demand calculated from the ISO13790 monthly method usually correlated well with dynamic simulations, *but the monthly values do not* as the seasonal storage effects are neglected. This results in higher heating values early in the heating period and lesser values later.

ECCHO assumes a fixed indoor heating set point of 20C and cooling set point of 25C to comply with Homestar. The custom calculation allows a user to input a different set point temperature. The heat balance is based on the difference between these assumed internal temperatures and the outdoor temperature data.

Building thermal mass can be set in the UI to custom, user defined or default values. These values follow the PHPP definition below and are in Wh/K/m^2. ISO13790 uses J/K/m^2 and ranges from 80,000 to 370,000 J/K/m^2, which corresponds to 22 to 103 Wh/K/m^2. This is a lower range than the PHI recommended range: a minimum of 60 Wh/K/m^2 for a fully

lightweight to a maximum of 204 Wh/(m2K) for a fully heavyweight building. (A fully heavyweight building would have exposed concrete slab/walls/ceilings with external insulation.)

ECCHO allows for either user-defined or pre-defined thermal mass parameters. The pre-defined values are as follows in Table 1 and are loaded from thermal_mass.csv. The overall thermal mass of the building can be estimated by considering each space in the building's construction mass for each of the six surfaces that make up that space and then weighting them for the floor area with each construction type. For example, if all the spaces in a building have one massive surface (the floor) then the overall thermal mass is 60+24 or 84 Wh/K/m^2. In the case of a two-level building where only the lower level has a thermally massive floor, then it is 84/2 + 60/2=72 resulting in 72 Wh/K/m^2. Thermal mass impacts the time constant of the building and how much of the gains can be counted to reduce the heating needed to maintain the temperature of the building.

Table 1: Building thermal mass based on constructions for structure.

Building frame and structure	Thermal mass Wh/K/m^2
Timber floor on piles	60
Concrete slab single level timber	84
Concrete slab two levels timber	72
Concrete slab and concrete midfloor	84

Gain utilisation factor

The monthly gains are adjusted by a gain utilisation factor. This dimensionless factor expresses the 'usefulness' of the heat gains in terms of the percentage of total gains available to contribute towards heating. It is a function of the thermal mass (heat capacity) of the building, the average losses and the ratio of gains and losses for each month.

The method is applied exactly as outlined in ISO 13790 with empirical constants aH,0 and tH,O as follows: aH,0 = 1, tH,O = 16

These constants are taken directly from PHPP and have not been adapted for New Zealand. They differ only slightly from the default constants in ISO 13790 which are aH,0 = 1 and tH,O = 15 for the monthly method.

There is an equivalent loss utilisation factor for cooling that is also as per ISO13790. The constants are the same as above for heating.

Python: Function heat_balance

Space cooling calculation – monthly method

Cooling balance

Annual cooling demand (kWh/year) is calculated in ECCHO by estimating, and then summing, the residual heat removal required to balance the gains and losses for each month. This is completely analogous to the heat balance and is a quasi-steady state method calculating the cooling balance over each month. The dynamic effects are considered by an empirically determined loss utilisation factor.

Python: Space cooling

Losses

Transmission heat losses

The total yearly transmission heat losses through external element areas are calculated as the sum of the monthly losses. Each monthly heat loss, kWh, is calculated as the product of the area, U-value, the difference between internal and external temperatures (i.e. the average monthly temperature) and the number of hours in each month.

Sum Ui x A x (inside temperature – average monthly outdoor temperature) x hrs per month

Function areas_calc3

This function inputs areas, windows, thermal bridges and returns areas_sum which contains the overall heat loss grouped as shown in PHPP Areas sheet at the top. This is simple summation.

Ground heat transfer

The ground heat loss (for slabs and suspended floors within 500mm of the ground) is calculated in ECCHO based on the methodology in ISO13370:2007, which calculates monthly ground temperatures below the slab. ECCHO then uses these monthly ground temperatures and the slab thermal transmittance, along with slab edge and floor-to-ground thermal bridge PSI values, to calculate the heat loss/gain from the ground. Note this is done using the methodology in PHPPv9.6a (the methods in PHPPv10 are different).

ECCHO calculates a ground temperature based on (1) the winter internal temperature (20C or custom) and (2) the summer internal temperature (25C or custom). Which ground temperature is used depends on the assumption of hours of heating or cooling in that month.

Sum Ui x A x (inside temperature – average effective monthly ground temperature) x hrs per month

Note several simplifications are assumed for the ground calculation. Ground thermal conductivity is assumed in ECCHO online to be fixed at 2 W/(mK) (the same as the NZBC) with a heat capacity of 2 MJ/(m³K). The crawl space is assumed to not have insulation on the walls (R0.4) or ground (R0.17) below the building. Crawl space height is assumed to be 0.8m with a local wind velocity of 4 m/s and a wind shield factor of 0.05, which is considered reasonable for an average suburban location. The ventilation area openings of the crawl

space are set to 0.35% of the foundation area (the NZBC E2/AS1 minimum of 3,500 mm^2 of opening per sqm of foundation area). Ground water correction assumes a 3m depth and a flow rate of 0.05 metres/day.

Python code

Ventilation losses

Heat losses from ventilation (both purpose and infiltration) are approximated following the methodology in ISO 13790. In this function the overall annual ventilation loss is then calculated as the sum of the monthly losses:

Ventilation loss per month = Effective_vent_rate * 1/3 * (indoor temp – avg monthly outdoor temp) x hours in the month. Note 1/3 is the volumetric heat capacity of air (i.e. 1200 J/ (m^3K) / 3600 s/hr).

The Effective ventilation rate is based on purpose provided ventilation rate, infiltration and heat recovery efficiency and provided in m^3/hr.

The ventilation volume is the CFA x average room height.

The purpose provided ventilation rate is calculated as the maximum of 0.35 air changes (based on the CFA x average stud height) or 7.5 litres/minute/occupant, whichever is highest in accordance with NZS4303. The occupancy used for this calculation depends on whether ECCHO is in Homestar or Custom mode. In Homestar mode ECCHO uses the default occupancy based on the floor area. In Custom mode ECCHO uses the custom occupancy set by the user.

The infiltration rate is calculated according to the approach in EN832 (also EN12831) based on the assumed airtightness of the home (either from a pressure test or the airtightness assumed from the age of the home). Note that the infiltration rate is reduced by the depressurisation provided by extract-only ventilation when compared to window-only intermittent ventilation. ECCHO Online assumes moderate protection and several sides exposed. These variables cannot be changed currently so are standardised.

WINDPROT-E and WINDPROT-F are constants representing the wind protection level as in Table 2 below taken from the Excel version of ECCHO.

Table 2: Wind protections coefficients. Note ECCHO set to assume moderate protection with several sides exposed.

Coefficient e for wind protection class:	several sides exposed	one side exposed
No protection	0.1	0.03
Moderate protection	0.07	0.02
High protection	0.04	0.01
High protection	0.04	0.01
Coefficient f	20	20

The effective heat recovery efficiency is zero for intermittent and continuous extract systems. For MVHR systems it is a function of the MVHR unit's specified heat recovery efficiency, adjusted for losses from ductwork and the location of the unit. This function calculates the heat recovery efficiency adjustment based on inside/outside installation and duct insulation. Duct diameter, insulation thermal conductivity and flow rate for the heat transfer calculation are hard coded.

Python code: Ventilation calculations

Gains

Solar gain through windows and opaque elements

The solar gain through windows and opaque elements is calculated as per ISO 13790:

$$\Phi_{sol,k} = F_{sh,ob,k} A_{sol,k} I_{sol,k} - F_{r,k} \Phi_{r,k}$$

Where the solar shading reduction factor, $F_{sh,ob,k}$ (REDFACSH in ECCHO) is assumed to be 0.7 for opaque elements and calculated separately for windows.

The solar irradiance $I_{sol,k}$ is taken from the monthly climate data for each cardinal for each location, adjusted for the orientation and tilt angle of each surface and then summed. This is directly a function of orientation and tilt angle and is not binned to the closest angle (as suggested in ISO 52016-1:2017 Section 6.6.8.2 Note 3, which recommends using discrete 45-degree sectors to avoid this adjustment for the specific orientation and tilt).

The conversion is based on dynamic hourly simulations that take the different parts of the solar radiation—direct, diffuse and ground reflected—into account. Using these calculations as a reference, PHI derived a trigonometric interpolation routine that is used in PHPP. This is a proprietary PHI algorithm that is not found in any standard. Note that as we are interpolating between the radiation values for the horizontal and the cardinal points, large errors are unlikely as the shape of the curves and the radiation changes with orientation and tilt are smooth.

ECCHO online for glazing uses the additional shading reduction factor PHPP defaults of 0.95 for dirt and 0.85 for non-vertical radiation incidence along with the 0.9 factor (a correction factor for non-scattering glazing as specified in ISO13890 for the solar transmittance of glazed elements).

Where the following constants are assumed in ECCHO online (except for special ASHRAE140 cases):

radiation shading factor REDFACSH = 0.7

external absorption coefficient EXTABS = 0.7

exterior emissivity EXTEM = 0.9

external surface resistance RE = 0.04 m2.K/W

external radiative heat transfer coefficient HRAD = 5 W/m2.K

external convective heat transfer coefficient HKON = 15 W/m2.K

This is a slight variation on the formula in ISO 13790.

Radiation to the sky

Radiative losses and gains to the sky from surfaces is calculated according to ISO13790. ECCHO uses sky temperatures for each month taken from climate data. The calculation includes a derivation of surface temperatures from the air temperature assuming a surface resistance of 0.04 m2.K/W.

$$\Phi_r = R_{se}\, U_c\, A_c\, h_r\, \Delta\theta_{er}$$

Python code: window calculations

Internal heat gains

Internal heat gains are set in the user interface as “default”, “nzs4218” or “building_code”.

In Homestar default mode, ECCHO calculates internal heat gains for the purposes of calculating a conservative winter annual heat demand. This is deliberately low and standardised. A common internal heat gain used residentially would be 5W/m^2, however this is based on historic data and does not consider modern energy-efficient lighting and appliances. It also unrealistically makes homes appear ‘self-heating’ in warmer climates, which does not reflect reality.

NZGBC expects that HEEP 2.0 from BRANZ will be able to update the default internal heat gains considering modern energy-efficient lighting and appliances.

In nzs4218 or building code mode, ECCHO calculates the internal heat gain based on nzs4218 values converted to a steady state value. This also accounts for a 100W heat gain from domestic hot water heating if the cylinder is located inside.

When in custom mode, the internal heat gains are calculated from first principles based on the custom occupancy and chosen appliances (note if occupancy is set to zero, so are the internal heat gains).

Python: internal heat gain calculations

Space overheating calculation

The overheating calculation uses the PHPP method, which calculates an annual temperature curve for a case without active cooling (heat recovery bypass function enabled and no heating). "The temperatures are then sorted by magnitude to produce an annual temperature/duration curve. A line of best fit in the vicinity of the temperature threshold of interest gives the percentage of time in which interior temperatures exceed the desired limit. This method allows determination of the acceptability of the interior temperature without hourly climate data, and with only a small number of entries." [PHPP 2015 pg. 183] This methodology was first developed in a working group [Feist 1999] and then later translated into PHPP.

Table 3: PHPP table of summer thermal comfort categories. Note the overheating prediction only roughly assesses the summer thermal comfort into these categories.

% time exceeded	Summer thermal comfort
h>25 °C	assessment
> 15%	catastrophic
10 – 15%	poor
5 – 10%	acceptable
2 – 5%	good
0–2%	excellent

Note the expected accuracy is only sufficient to sort into categories as shown in Table 3 above. Experience (Schnieders 2012) with dynamic models have shown that climatic differences from one summer to the next can result in around 5% shift in overheating and thus a more accurate method may not be more predictive. "This method is not appropriate to apply to single rooms except to see which rooms may contribute most to overheating– this model approach is based on a dynamic single zone building" [PHPP 2015]. This method is also not appropriate for buildings with poor thermal performance where the single day load exceeds that which can be stored in the building thermal mass.

In order to be able to show the influence of individual hot days as well, and to obtain meaningful results for small overheating frequency values, the month of July is

additionally divided into several parts: the cooling load day at the end of the month, the four preceding days with slightly lower temperatures and radiation values, another twelve preceding days with lower temperatures and radiation values once more, and the then the rest of the month. In the process, the month is divided in such a way that the monthly average values for the outdoor temperature and solar incidence for July remain unchanged. The average values for the indoor temperature are also determined in the same way for these shorter periods.

The frequency of overheating can now be ascertained from all average values and the respective time periods. Although the temperatures determined using the simplified method differ greatly from the hourly simulation at times, the overheating frequencies correlate exactly for both methods to enable classification in accordance with *Table 3*. [PHPP 2015].

Note the code in this function calculates the overheating in the same manner as PHPP. Also note that July is cited as the hottest month as the climate data is shifted six months to simulate the northern hemisphere. The methodology is laid out in Feist 1999.

Function summer_overheating

This function inputs the vent_data, dwelling_data, location_climate_data_df, windows_sum, areas_rad, areas_sum, vent_calc, custom_ihg, window_airchange, location_climate_data, internal_heat_gain. The function then returns the losses_gains, gain_loss_calc with the overheating data. These outputs are not used in the heating or cooling balance.

Summer ventilation and impact on overheating

ECCHO online calculates the summer additional air change rate based on window openings. This implements simplified calculations from the PHPP Summer Ventilation sheet 'Estimation for window air change rate' (not the night ventilation rate), assuming no height difference between the openings, an average temperature difference of 4K and a wind speed of 1 m/s. The resulting air change rate is used in the overheating calculation only to reduce overheating value. It is important to realise this is an average impact at best and hot still days will exceed the values.

Opening time is set in the UI with a drop-down menu and directly impacts the ventilation rate.

Function sumvent

This function calculates summer airchange rate based on window openings. Inputs are vent_data, tfa (from dwelling_data) and the function returns the resultant_air_change which is used in the overheating calculation only. Note although the windows are assumed to be on the same level (no stack effect between windows) the temperature difference is assumed to cause ventilation across the height of the window itself.

Note 'win_security' is a numeric set in the UI to correct for time windows are open.

Python code

Calculation of R/U-values assemblies

ECCHO includes an assembly R/U value calculator. This follows the method in ISO6946: 2017 for thermally inhomogeneous layers. The external and internal surface thermal resistances are varied slightly from that in ISO 6946 based on the climate zone being predominately heating or cooling dominate. Note that the change of surface resistances in PHPP is a PHI change from ISO6946 methodology, based on the idea that the predominate energy flow direction reverses in cooling predominate climates. This is not in accordance with the ISO standard and there is some agreement at PHI to put this back to standard for slab-on-grade.

This U-value calculation method is only able to handle three sections each with one percentage through the entire construction. If there are more sections than this, another calculation tool should be used. The most accurate method would use a 2D finite element tool such as Flixo or Therm to ISO10211.

The U-value calculation follows ISO6946 approach for thermally homogeneous and inhomogeneous layers but does not calculate tapered layers or calculate corrections for metallic fasteners. Note that this method is not valid for cases where the upper limit of thermal resistance to the lower limit exceeds 1.5x or where insulation is bridged by metal. If this limit is exceeded the thermal resistance should be calculated using ISO10211 and entered as a custom U-value or R-value. The methodology from ISO6946 is to calculate the upper and lower limit of the thermal resistance and then use the average. Note for reference the upper limit is the NZS4214 calculation method (with slightly different surface resistances).

Python code

Window R/U-values calculations

The window U-value is calculated using the approach in ISO10077-1 with a single frame width per window and a single frame type. To have a comparison to 2023 H1 Code, the window U-value is also calculated without the thermal bridge losses due to the installation in the wall. This calculation takes as input the glazing type and frame type, the height and width of the window and how many sides of the window are to other windows rather than to the wall. This is very important as the losses due to the window being installed in the wall are set to zero if it is adjacent to another window. Note that the window orientation and angle set from the areas they are installed in. The windows are grouped by orientation or if the angle to horizontal is less than 30 degrees, into horizontal windows. This puts all windows into one of these five groups (four cardinal directions plus horizontal).

Python code

Occupancy

The standard (default) occupancy originally used in ECCHO for Homestar compliance was the same as that used in PHPP for Passive House compliance. This in turn was based on some UK/German research into actual occupancy in homes in those countries. A default occupancy is used for the calculation of ventilation rates, hot water consumption and summer indoor heat gains when calculating energy demand for Homestar. A custom occupancy can be inputted. This is only used in 'custom' mode.

Occupancy is tied to conditioned floor area (CFA, not bedroom numbers). Originally this was based on PHPP as follows:

Number of occupants = 1 + 1.9 * (1 - EXP(-0.00013 * (CFA-7)^2)) + 0.001 * CFA

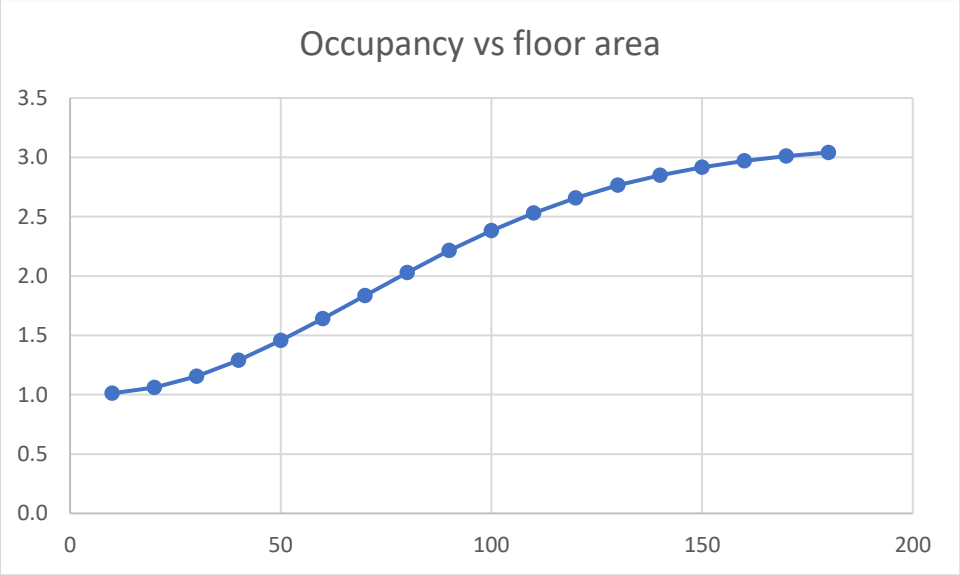

Figure 2: PHI default occupancy versus floor area used in ECCHO.

The main calculations impacted by occupancy are internal heat gain and hot water demand. For default Homestar internal heat gain however, this is limited to overheating/cooling. Internal heat gain for winter heat demand is also standardised based on floor area as

IHG (Watts) = 2.1 * CFA + 50

The main area of impact therefore is hot water demand. The PHPP occupancy algorithm effectively caps occupancy at around 3.1 people per home meaning that as homes get larger the same amount of hot water gets progressively divided by a larger floor area which in turn makes the electricity demand targets easier to meet (which further means less incentive to improve hot water efficiency).

NZGBC has obtained StatsNZ data on occupancy of New Zealand homes based on census data. This is illustrated in the graph below which also compares it with the current ECCHO (PHPP) assumed occupancy.

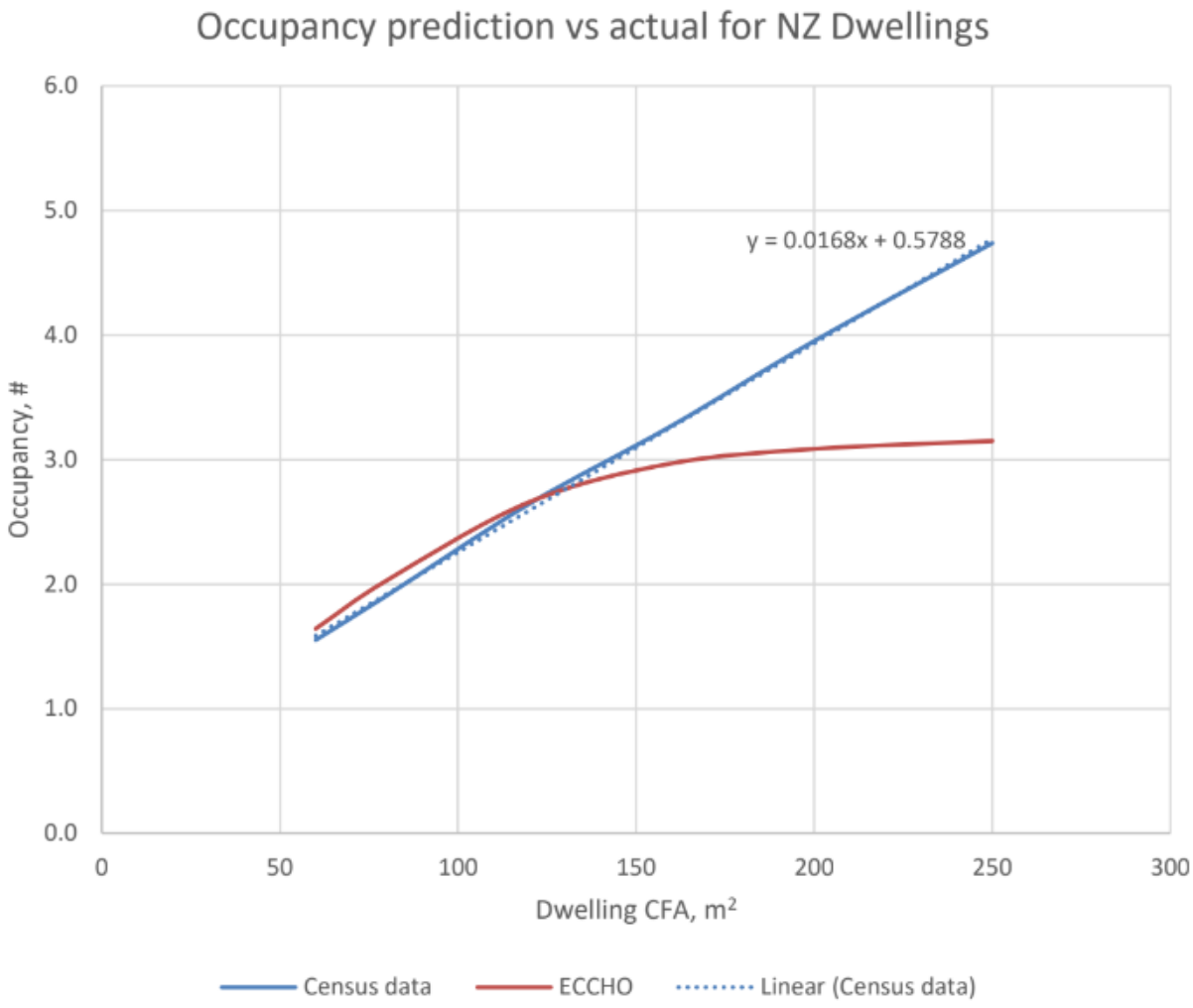

Figure 3: Occupancy versus CFA for NZ. Source NZGBC ERP review Dec2023.

As the graph shows, for the smaller homes (50 – 120m2), the existing algorithm is broadly aligned with reality. As homes get larger, however, the actual occupancy is likely to be higher, diminishing the impact of hot water efficiency on electricity demand in these larger homes.

One possible justification for keeping the occupancy algorithm as it is would be to keep consistency with Passive House calculations. However, as noted, we have not proposed to change the internal heat gain algorithm meaning that the main heat demand calculation would remain consistent. The main area of departure is in the calculation of hot water demand as we do not consider the current algorithms to accurately represent larger homes.

As such ECCHO has changed the occupancy algorithm to the following linear equation with version 3:

Number of occupants = 0.0168 * CFA + 0.5788

Python code

Climate data

Weather files are taken from PHPPv9.6a and this data is based on the NIWA hourly climate files with some minor adjustment by PHI. Meteonorm can be used to generate the monthly climate data from hourly EPW files, except for the heating and cooling load columns. The columns with monthly data are measured data summed for the month. This climate data was checked for heat loss across the available New Zealand weather stations here: sustainableengineering.co.nz/phclimatenz

The heating and cooling load data is generated and is not a measured climate parameter. Rather, the heating and cooling load data is produced by PHI using a dynamic hourly building simulation model (DYNBIL) designed specifically to model Passive House projects. In essence, DYNBIL dynamically simulates an example Passive House structure adjusted to just meet the certification standard on SSHD for the provided hourly climate file (Schnieders, 2003).

The heating and cooling load data is produced by looking at the rate of change of heat transfer. This process requires a Typical Meteorological Year (TMY) hourly weather file that has been checked for applicability to the region in which the loads will be used. NIWA had developed TMY hourly files to represent recommended New Zealand regional climates (Liley, Sturman, Shiona & Wratt, 2008), but these have not been checked beyond that documented in Quinn 2015.

The monthly temperatures are adjusted to account for the difference between the height above sea level (ASL) of the weather station for which the weather data was obtained and the height ASL of the home being assessed. ECCHO applies a reduction/increase of 0.6C for every 100m increase/decrease in the height ASL of the building above the height ASL of the weather station.

These adjusted monthly temperatures are used to return the climate type that is used to set the surface air film resistances. The determination of climate 'type' is carried out by summing the number of heating predominant months (average temperature less than 10C) and the number of cooling predominant months (average temperature greater than 20C).

A climate is considered 'heating predominant' if the number of heating months is greater than the cooling predominant months + 2.

A climate is considered 'cooling predominant' if the number of cooling months is greater than the heating predominant months + 2.

Otherwise, the climate is considered 'temperate'.

Python code

Orientation of building and site to climate

The climate data and equations in ECCHO assume that the sun tracks to the south since the reference standards have generally been written for Europe. Southern hemisphere climate data must therefore be shifted by six months and the orientation of all areas mirrored about the equator so that north walls face south and vice versa. Note that the Climate_data.csv file already has the data mirrored about the equator as it is in PHPPv9.6a. Also note that the monthly data is simply shifted six months and as the number of days in the month is not shifted the climate data is slightly changed. For example, the southern hemisphere warm February with 28 days is shifted to August with 31 days. This does impact the thermal predictions but only slightly and was accepted in the development of PHPPv9.6a.

Python code

Fuel demand and carbon emissions

Overview

The overall fuel demand for a home is broken down into energy used for:

- space heating (discussed previously)
- hot water
- custom appliance annual energy demand
- lighting
- auxiliary annual demand for fans and pumps

The carbon emissions are calculated from the fuel demand multiplied by the carbon intensity of the fuel plus the carbon intensity of the refrigerant leakage.

Hot water

Hot water demand is estimated in ECCHO from the total hot water draw off from taps and fittings plus any hot water losses from hot water cylinders (if present) and pipework.

The total energy required to heat the total hot water demand is estimated based on the average cold water temperature for the home's location. The cold water temperature is assumed to be the average outdoor temperature for the house location + 1C.

The total draw off from taps and fittings is estimated from the building's occupancy (see building occupancy section) and the amounts are pulled from hot_water.csv. The appliance DHW demand is added to this.

Table 4: DHW usage from hot_water.csv.

Application	Calculated per person?	Time of use per use, minutes	Amount of uses according to type of use,	Flow rate, litres per second	Useful temperature , C
Shower	Yes	6	0.9	12	38
Hand wash basin	Yes	0.3	3	4	30
Bathing	Yes	10	0.03333	15	38

Teeth brushing	Yes	0.05	2	4	30
Cooking/ drinking	Yes	0.25	1	6	45
Dishwashing	No	0.6	1	6	45
Cleaning the kitchen	No	0.5	1	6	38
Cleaning rooms	No	1	0.142857143	6	38

The annual kWh for generation of hot water is then calculated from the daily demand and the temperature difference.

Hot water storage losses are calculated for the tank storage and pipe losses for DHW in a similar manner to PHPP. Note that currently the storage cylinder is hard coded to 60C, hot water flow temperatures to 55C, external pipe diameter to 0.015m and 6 tap openings per day.

Any annual cylinder losses are derived from AS/NZS 4692.2, which specifies maximum heat losses, corrected for indoor/outdoor temperature depending on the location of the cylinder (indoors or outdoors). The following table sets out the unit heat loss for cylinders between 15 and 400 litres in capacity:

Table 5: Hot water cylinder losses in W/K for range of sizes in litres.

Cylinder size (litres)	Heat loss W/K
0	0
15	0.46
25	0.5
45	0.58
50	0.6
90	0.76

135	0.94
180	1.12
225	1.3
250	1.4
270	1.48
300	1.6
370	1.88
400	2

Cylinder losses assume a new cylinder insulated to the current NZ Standard (NZS 4305:1996). It is not currently possible to change this default to a more poorly performing older cylinder. Losses are calculated assuming a cylinder temperature of 60C. The deltaT is taken as being between this temperature and the specified indoor temperature (20C by default for Homestar) or the average outdoor annual temperature.

Pipe losses (W/K) are based on an assumed pipe length, which is calculated as follows:

pipe_length (m) = Conditioned Floor Area^(1/3) * tapping_points + 10

where tapping points = 1+CFA/50.

ECCHO assumes six tap openings per person per day. Therefore, a volume of hot water is assumed to be drawn into the total length of pipework that then cools down from the delivered temperature (55C) to the indoor temperature six times per day per occupant for the full length of pipe. This is the same calculation methodology as used by PHPP.

An additional loss is then included for the fittings and 2m of pipework off the top of any cylinder (again, if present). This is based on the extent to which these are insulated. The default ('normal') is a continuous loss of 0.5 W/K (deltaT being the difference between the cylinder temp and the indoor/outdoor cylinder location) with 0.25 W/K for medium insulation and 0.1 W/K for excellent insulation. See *Guide to ECCHO* for details of what each level represents.

Python code

Appliance energy demand

ECCHO calculates the energy demand from appliances including both appliance hot water and electricity. Only the cooktop has an option for LPG or natural gas (ie a gas oven etc is not an option).

There is the option in ECCHO to include a domestic hot water connection to dishwashers and washing machines. This will increase the overall domestic hot water demand, but potentially reduce overall dwelling electricity demand if the domestic hot water heat source has a COP greater than 1.0. This is because the default assumption is that these appliances heat hot water with a resistive heater internally.

Python code

Lighting

ECCHO assumes that each light is on for an average of 2.2 hours, 1.2 hours and 0.7 hours per day respectively in the main living areas, bedrooms and all remaining areas. This is an average of summer and winter daily hours of use taken from the BRANZ HEEP work and corroborated by Australian analysis. Note that whole rooms may be lit for longer than this in practice, but this is the average for all lights in each room, taking into account that not all lights may be on at once in larger rooms like living/dining areas that have multiple circuits.

Python code

Energy from fans (and pumps)

This is the overall auxiliary annual demand for fans and pumps per unit reference area. The definition of auxiliary energy is the same as PHPP and includes the energy necessary to run or control the mechanical systems: heating, ventilation and DHW (which would include any solar thermal system power demand). ECCHO does not currently calculate any energy from pumps used in the dwelling. This because central heating is relatively rare in the New Zealand market. This could be added if needed.

Python code

Refrigerant leakage

Refrigerant leakage is modelled as a continuous leakage rate over the service life of the system plus an assumed end-of-life loss. These loss rates are shown in the table below. This

conservative position is due to the lack of good refrigerant recovery data in New Zealand and these numbers can and should be updated once more research is available. This does result in rewarding lower GWP refrigerants, with one such as propane performing quite well.

Table 6: Refrigerant properties from refrig_properties.csv. Note leakage rate, end of life loss and GSP

Refrigerant	ODP	GWP	Leak rate	End-of-life loss	Use case	Source	Comment
HFC-134a	0	1300	7.0%	10%	replaced CFC-12	Calm and Hourahan, 2011	WMO 2014, Appendix 5A
HFC-32	0	677	7.0%	10%	alternate to R-410A for unitary	Calm and Hourahan, 2011	WMO 2014, Appendix 5A
HFC-404A	0	3943	7.0%	10%	low temp applications; replaced CFC-502	Calm and Hourahan, 2011	IPCC AR5
HFC-407C	0	1624	7.0%	10%	HCFC-22 replacement for DX unitary	Calm and Hourahan, 2011	IPCC AR5
HFC-410A	0	1924	7.0%	10%	new equipment replacing HCFC-22	Calm and Hourahan, 2011	IPCC AR5
HFC-417A	0	2300	7.0%	10%	DuPont ISCEON@MO59 repl HCFC-22	Calm and Hourahan, 2011	IPCC AR5

Python code

Glossary

CFA	Conditioned Floor Area
IHG	Internal Heat Gain
TFA	Treated Floor Area
UI	User Interface
GWP	Global Warming Potential – usually in kgCO2e over 100 years

References

Quinn, Jason E. (2015), "Affordable certified climate files–process used in New Zealand," South Pacific Passive House Conference (SPPHC) 2015.

Schnieders, J. (2003) Climate Data for the Determination of Passive House Heat Loads in Northwest Europe. Passive House Institute. Project sponsored by the European Commission under contract EIE-2003-030, PEP.

Liley, J. B., Sturman, B., Shiona, H., and Wratt, D. S. (2008). Typical Meteorological Years for the New Zealand Home Energy Rating Scheme. NIWA Client Report: LAU2008- 01-JBL, November 2008

PHPP (2015) Passive House Planning Package version 9 Passive House Institute, 2015

Roulet, C A, and B. Anderson. 2006. "CEN Standards for Implementing the European Directive on Energy Performance of Buildings." In PLEA2006 - 23rd Conference on Passive and Low Energy Architecture, 1–6. Geneva.

Schnieders, J. (2012) PB 41: Planning tools for the summer situation in non-residential buildings, PHI, Passipedia. https://passipedia.org/phi_publications/pb_41/planning_tools_for_the_summer_situation_in_non-residential_buildings

[Feist 1999] Feist, Wolfgang (Hrsg.): „Passivhaus Sommerfall"; Protokollband Nr. 15 des Arbeitskreises kostengünstige Passivhäuser, Passivhaus Institut, Darmstadt 1999 ("Passive House Summer Case"; Protocol Volume No. 15 of the Working Group for Cost-Efficient Passive Houses; Passive House Institute, Darmstadt 1997). Primary article here https://passipedia.org/phi_publications/pb_15/a_simplified_method_for_determining_thermal_comfort_in_summer_for_buildings_without_active_cooling

Appendix Code

Database relationships in ECCHO

ECCHO uses several databases to store the information for inputs and outputs of the computer model. The user only interacts with these databases via the User Interface (UI) but it can be useful to understand the overall structure of the database to understand how best to use the code.

1. At the top level are Users.
    a. Glazing and Framing tables belong to Users. They can be used across multiple projects
2. Each user has multiple Projects.
    a. Assemblies are children of Projects. They can be copied in the UI from project to project
        1. Layers. This is a table of individual layers of an assembly (plasterboard, timber, insulation etc). Child of Assemblies.
3. Each project has multiple Dwellings. Dwelling data includes miscellaneous data such as the number and size of hot water cylinder.
4. Each Dwelling has:
    a. Multiple Areas (walls, roofs, Floors – see below)
        1. Windows are a child of Areas
        2. Windows have a many-to-many relationship to Glazing and Framing
    b. Multiple Floors (separate table since these are sent to the ground temperature calculation). ECCHO duplicates a floor in Areas and Floors. The former is used for the annual heat loss calc, the latter for the ground temperature.
    c. Ventilation. Data on the ventilation systems used in each dwelling.
    d. Thermal bridges
    e. Space heaters
    f. Hot water heaters
    g. Showers
    h. Lighting
    i. Appliances

j. Refrigerants

There is also a separate MVHR table that has no parent/child relationship. It is simply a record of all the ventilation systems we have in ECCHO. It iis slightly misnamed as continuous extract systems are also contained here. Data cannot be updated by users but must be done through the backend by NZGBC. This is a check on the quality of data on ventilation systems in ECCHO.

Basic dwelling information

Class Dwelling_calculated (see dwelling.py)

This is the main dwelling object class definition (note capital D in Dwelling_calculated). This contains all the data (in the form of Pandas dataframes) for calculating the energy demand of the dwelling. The dwelling object's parameters are all pandas objects (tables of data). Tables created are: project_data, dwelling_data, appliances, assemblies, areas, windows, thermal_bridges, gnd_temp_data, vent_data, lighting_data, hot_water_data, shower_data, space_heater_data, refrigerant_data, mvhr_units.

This object contains the custom method calculate_energy that runs through the calculations for a dwelling.

A more detailed list of the data class for each object used is in models.py

Module models.py

Contains classes with the database structure – contents summarized below.

class User	*Top level are Users. Note that Glazing and Framing tables belong to Users. They can be used across multiple projects*
class Project	*Defines fields for the database entries for a Project class which can contain multiple dwelling and assemblies*
class Dwelling	*Defines fields for the database entries for each dwelling which contains multiple areas_, windows_, grounds_,…, This also generates the unique ID*
class Glazing	*Defines fields for the database entries for description, U_glazing, g_glazing; relation with Window_data to allow backref of glazing*
class Framing	*Defines fields for the database entries for description, U_frame, frame_width, frame_psi, frame_install_psi; relation with Window_data to allow backref of framing. Note only single framing width, single PSI_g== frame_psi, single PSI_install==frame_install_psi*

class Assemblies	*Defines fields for the database entries for description, ass_type, adjacent, u_value, percentages for the layers; then links to data to/from areas, grounds, layers; Note hard coded three fractions for layers.*
class Layers	*Defines fields for the database entries for each layer taking in name, Rswitch, lambda for each of three columns and then the single layer thickness*
class Areas	*Defines fields for the database entries similar to Areas sheet in PHPP columns*
class Room	*Defines fields for the database entries for HHS room calculations. Heater size, volume etc.*
class Window_data	*Defines fields for the database entries similar to windows plus shading sheet in PHPP only overhang depth/height and then shading % winter/summer*
class Ground	*Defines fields for the database entries area_m2, p_length, assembly_id and links to assemblies*
class Ventilation	*Defines fields for the database entries similar to ventilation sheet but limited to fewer inputs*
class Th_bridges	*Defines fields for the database entries*
class Space_heaters	*Defines fields for the database entries*
class Hot_water_systems	*Defines fields for the database entries*
class Showers	*Defines fields for the database entries*
class Mvhr_units	*Defines fields for the database entries*
class Lighting	*Defines fields for the database entries*
class Appliances	*Defines fields for the database entries*
class Refrigerant	*Pulls in refrigerant GWP and system life along with leakage rates and sets emissions*
class Comments	*Tracks comments and resolved plus date.*
class Variants	*Stores fields from online table inputs and results.*

Functions

App/Calculation/		
Areas.py		
	areas_calc1	rotates the building by adding the user entered orientation in degrees, makes sure all angles between 0 and 360; Then flips to northern hemisphere. if float(latitude) < 0: areas["dev_nth"] = areas["dev_nth"].apply(lambda x: 180 - x if 180 - x >= 0 else 180 - x + 360) Returns areas
	areas_calc2	calculate solar load through opaque surfaces. Pulls in areas, climate, windows, project_data, dwelling_data and then Returns areas_rad
	areas_calc3	calculate overall heat loss. Returns area_sum
Climate.py		
	location	read in climate file and set location
	climate_data	adjusts temperature data by altitude lapse factor multiplied by the location height_above_sea - climate file location height above sea level. Climate calculations used elsewhere including peak day etc for overheating
	project_climate_type	defines heating or cooling predominate or temperate climate based on climate data. Months <10C heating, >20C cooling
Cooling.py		
	cooling_balance	main cooling demand calculation (based on PHPP cooling worksheet).
Ground.py		
	gnd_temp	Ground sheet calculations PHPPv9.6a; Hardcodes lots of options from Excel; Only 'Slab on grade' or 'Susp floor' but does allow multiple areas.
Heating.py		
	heat_balance	main heating demand calculation (based on PHPP heating worksheet)
	heat_balance_graph	back calculate contribution of different areas (walls, floor, windows etc) to heat gains and losses for heat balance graph

Shading.py		
	shade_calc_winter	
	shade_calc_summer	
Summer.py		
	summer_overheating	main overheating calculation (based on PHPP summer worksheet)
Uvalues.py		
	calculate_u_value	U_values calculated according to ISO 6946
Ventilation.py		
	vent_calc1	calculate effective air change rate based on infiltration, heat recovery efficiency etc; hard codes ventilation protection factors
	effective_heat_recovery_eff	calculate heat recovery efficiency adjustment based on inside/outside install and duct insulation; Hard codes duct diameter, insulation and flow rate for heat transfer calc
	sumvent	calculate summer air change rate based on window openings
Windows.py		
	win_U_value	Calculates window U value and glazing percentage from glazing and framing data
	win_calc	initial lookup of orientation based on orientation of wall installed in, plus assign to window cardinals.
	win_calc2	calculate global radiation on the surface of windows
	win_calc3	
	win_calc4	sum all installed windows to cardinals
__init__.py		Pulls in dwelling and U-values for the version of the calculation engine
aux_lighting.py		
	lighting_calc	calculate overall annual lighting energy demand from lighting data
	aux_calc	calculate overall auxiliary annual demand for fans and pumps
	custom_appliances	calculate annual demand for energy from appliances

	ihg	calculate internal heat gains from custom appliances in dwelling Used for summer overheating and custom winter thermal demand.
dwelling.py		
	Dwelling_calculated	Main dwelling object. This contains all the data (in the form of Pandas dataframes) for calculating the energy demand of the dwelling.
	calculate_energy	dwelling custom method takes in dwelling data and calculates energy demand. Pulls in climate_data sorts the appliances then runs through areas_calc1, win_calc, win-calc2, win_calc3, win_calc4, areas_calc2, areas_calc3, gnd_temp, vent_calc1, sumvent, lighting_calc, custom_applicances, hot_water_calc, hot_water_storage, total_hot_water_demand, aux_calc, ihg, heat_balance, cooling_balance, summer_overheating, heat_balance_graph
hot_water.py		
	hot_water_calc	
	hot_water_storage	

Implementation in Python code

Method **calculate_energy** (see dwelling_py)

This dwelling custom method takes in dwelling data and calculates energy demand. Pulls in climate_data sorts the appliances then runs through areas_calc1, win_calc, win-calc2, win_calc3, win_calc4, areas_calc2, areas_calc3, gnd_temp, vent_calc1, sumvent, lighting_calc, custom_applicances, hot_water_calc, hot_water_storage, total_hot_water_demand, aux_calc, ihg, heat_balance, cooling_balance, summer_overheating, heat_balance_graph.

There is no physics inside of this method other than setting total_hot_water_demand = hot_water_demand + storage_losses + pipe_loss.

Go back

Function **heat_balance** see heating.py

This function takes as inputs the dwelling_data, location_climate_data_df, windows_sum, areas_rad, areas_sum, vent_calc, Ls, internal_heat_gain, custom_ihg and returns the gain_loss_list which contains the gain_loss_calc and gains_losses tables.

The function then calculates the heat balance in the same manner as the PHPP file (see hidden rows 90 to 133).

Function heat_balance_graph see heating.py

This function back-calculates the contribution of different areas (walls, floor, windows etc) to heat gains and losses for the heat balance graph from the totals. This means the amounts add up to the monthly method results from the heat_balance function. This function returns the loss_graph and gain_graph.

Go back

Function cooling_balance

The annual cooling energy demand calculation is based on ISO13790. This function inputs the vent_data, dwelling_data, location_climate_data_df, windows_sum, areas_rad, areas_sum, vent_calc, Ls, custom_ihg, window_airchange, location_climate_data,

internal_heat_gain and returns the losses_gains, and gain_loss_calc. Note at this time there is no cooling balance graph in ECCHO online.

Go back

Function and_temp

This function inputs dwelling_data, location_climate_data_df, gnd_temp_data and returns a modified location_climate_data_df, Ls, gnd_temp_data.

In order to implement the methods, the constants recommended in PHPP as defaults are used.

```
GND_LAMBDA = 2.0
    GND_HEAT_CAP = 2.0
    PER_PEN_DEPTH = math.sqrt(365 * 24 * 3600 * GND_LAMBDA / (math.pi * GND_HEAT_CAP * 1000000))
    UVALCRAWL = 5.9
    CRAWLHEIGHT = 0.8
    UVALCRAWLW = 2.5
    WINDVEL = 4.0
    WINDSHEILD = 0.05
    GND_WATER_DEPTH = 3.0
    GND_WATER_FLOW = 0.05
```

Note the crawl ventilation area is hard coded to NZBC E2/AS1 minimum of 3,500 mm^2 of opening per sqm of foundation area it is not conservative for heating if larger openings are used.

```
crawlventarea = 3500 / (1000**2) * gnd_temp_data.loc[x, 'area_m2']
```

These are then used with the location_climate_data_df to calculate the heating days and the length of the heating season along with monthly temperatures to sinusoidal temperature delays from ISO13370. The calculated phase_shift_T, and equivalent thickness and characteristic dimension is used for each floor are ('Slab on grade' or 'Susp floor') to calculate the ground temperature for that floor area.

These temperatures are then combined into monthly ground temperatures for summer, winter and the average (there are differing ground temperatures as the interior temperatures are assumed to differ between winter and summer) and added to the location_climate_data_df.

Go back

Function vent_calc1

This function calculates the effective air change rate based on infiltration, heat recovery efficiency etc; hard codes ventilation protection factors. Inputs required are the dwelling_data, vent_data, location_climate_data, mvhr_units and the function returns vent_calc. Note the air volume is the CFA x room height. In the code this is

```
vent_calc['Air volume'] = vent_data.loc[0, 'room_height'].sum() * dwelling_data.loc[0, 'building_TFA'].sum()
```

The purpose provided ventilation rate is calculated per:

```
vent_calc['derived_vent_rate'] = max(0.35, (dwelling_data.loc[0, 'occupancy'] * 7.5 * 3.6) / vent_calc['Air volume'])
```

The infiltration rate is calculated with WINDPROT-E and WINDPROT-F are constants in ECCHO online

```
vent_calc['inf'] = vent_data.loc[0, 'air_change'].sum() * WINDPROTE / (1 + WINDPROTF / WINDPROTE * (excess_extract / vent_data.loc[0, 'air_change'].sum()) ** 2)
```

Where the excess extract rate is either zero when using intermittent ventilation or MVHR or set to the continuous extract rate if using continuous extract ventilation. The result of this is that infiltration is reduced by the depressurization provided by extract only ventilation.

The effective ventilation rate (used to calculate heat losses) is then the sum of the purpose provided ventilation rate (adjusted for heat recovery efficiency) and the infiltration rate as follows:

```
vent_calc['effective_ach'] = vent_calc['derived_vent_rate'] * (1 - HReff / 100) + vent_calc['inf']
```

Function effective_heat_recovery_eff

Inputs are dwelling_data, vent_data, location_climate_data, mvhr_units and the function returns the effective heat recovery efficiency HReff.

The following inputs are set in the code as constants:

```
DUCT_DIA = 150
  INS_COND = 0.04
  LOFT_TEMP = location_climate_data.loc[0, 'winter_avg'] #assumed temperature in of space outside thermal envelope, eg loft in winter
  DUCT_VEL = 2.5
  NUSSELT = 70
  FLOW_RATE = 150 ##m3/h typical for a home
```

The heat flow in the duct is assumed to be turbulent flow to allow use of the Nussel approximation to estimate the duct surface temperature and resultant heat flux. The reduction in air temperature impacts the MVHR efficiency. Note efficiency of the MVHR is measured per PHI.

Go back

Function space cooling

Function win_calc2

This function takes as input the location_climate_data_df and pre-calculates the climate data table and returns it. This is only a function of the surface ALBEDO = 0.106 and the data in the climate file.

Function win_calc3

This takes in the location_climate_data_df and precalculated climate data from win_calc2 and windows table to calculate the radiation on the windows and returns the window radiation with and without shading for winter and summer returning windows_rad. This calls the Functions shade_calc_winter and shade_calc_summer passing in windows_rad and absolute value of the latitude.

Function win_calc4

Sums all installed windows to cardinals

Function shade_calc_winter

This function takes in the windows_rad table and the latitude and returns windows_rad with the windows_rad["shade factor"].

Function shade_calc_summer

This function takes in the windows_rad table and the latitude and returns windows_rad with the windows_rad["shade factor_summer"]

Function win_calc4

Groups and sums the data from windows_rad and calculates the summary data for these groups. Function inputs windows_rad and outputs windows_sum.

This uses the PHPP defaults of 0.95 for dirt and 0.85 for non-vertical radiation incidence.

windows_sum['Solr irr red ftr'] = windows_sum['Shade'] * windows_sum['Dirt'] * windows_sum['Non vert rad inc'] * windows_sum['Glz %']

windows_sum['Solr irr red ftr_summer'] = 0.9 * windows_sum['Shade_summer'] * windows_sum['Dirt'] * windows_sum['Glz %']

Note the 0.9 factor is a correction factor for non-scattering glazing as specified in ISO13890 for the solar transmittance of glazed elements.

Function areas_calc2

This function inputs areas, climate, windows, project_data and dwelling_data and then copies areas table into areas_rad and calculates the solar gain through the opaque elements returning the values as areas_rad.

The solar aperture is calculated as follows (ISO13790 annex E):

```
areas_rad.loc[(areas_rad["u_value"] != 0), "Solar apt"] = (
        1 / (EXTEM * HRAD + HKON)  / (1 / areas_rad["u_value"] - RE + 1 / (EXTEM * HRAD + HKON))
        * EXTABS * REDFACSH  * areas_rad["area_m2"] )
```

Go back

Function internal_heat_gain_calc (see main_calc.py)

This function inputs the dwelling_data and variable custom and checks if it is “default”, “nzs4218”, “building_code” or not set. This then returns the internal_heat_gain as a value or None if custom was not set.

In Homestar default mode ECCHO calculates internal heat gains for the purposes of calculating winter annual heat demand as follows:

```
internal_heat_gain = 2.1 * dwelling_data.loc[0, "building_TFA"] + 50
```

For nzs4218 or building_code mode ECCHO calculates the internal heat gain based on nzs4218 values converted to a steady state value. This also accounts for a 100W heat gain from the DHW if the cylinder is located inside.

```
if dwelling_data.loc[0, "building_TFA"] > 50:
    internal_heat_gain = 4.1 * dwelling_data.loc[ 0, "building_TFA" ] + 138.6 * 3 / (7 * 24)
else:
    internal_heat_gain = 4.1 * dwelling_data.loc[0, "building_TFA"] + 138.6 * (
      150 / dwelling_data.loc[0, "building_TFA"]) / (7 * 24)
internal_heat_gain = ( internal_heat_gain + 100 / dwelling_data.loc[0, "building_TFA"]
    if dwelling_data.loc[0, "cyl_location"] == "Inside"
    else internal_heat_gain)
```

When in custom mode the internal heat gains are calculated from first principles based on the custom occupancy and chosen appliances as follows (note if custom_occupancy set to zero so is custom_internal_heat_gain).

Function ihg (see aux_lighting.py)

This function inputs the appliances, location_climate_data_df, dwelling_data, pipe_gains, lighting_demand and then returns the custom_internal_heat_gain. This accounts for DHW impacts and appliance water usage impacts.

```
custom_internal_heat_gain = max(2.1 * dwelling_data.loc[0, 'building_TFA'] + 50,
        appliances['internal heat gains'].values.sum() + lighting_demand / 8760 + dwelling_data.loc[0, 'custom_occupancy'] * 100 / 8.76)
  if dwelling_data.loc[0, "custom_occupancy"] == 0:  custom_internal_heat_gain = 0
```

Go back

Function sumvent

This function calculate summer air change rate based on window openings. Inputs are vent_data and tfa (from dwelling_data); the function returns the resultant_air_change, which is used in the overheating calculation only. Note although the windows are assumed to be on the same level (ie no stack effect between windows) the temperature difference is assumed to cause ventilation across the height of the window itself.

Note 'win_securrity' is a numeric set in the UI to correct for time windows are open.

```
  g = 9.81 #accel due to gravity
  wind_vel = 1 #m/s
  avg_temp = 298 #K
  temp_diff = 4 #K
  dim_less = 0.02 #Volume flow rate due to wind pressure
  fD = 0.7
  Cd = 0.61
  DCp = 0.3
  for x in range(1,4):
    if x == 3:
      if vent_data.loc[0, 'win1_width'].sum() > 0 and vent_data.loc[0, 'win1_width'].sum() > 0:
        clear_opening.append(1 / (1 / clear_opening[0] ** 2 + 1 / clear_opening[1] ** 2) ** 0.5)
        vol_flow_thermal.append(0) #assume no height diff between window groups
        vol_flow_wind.append(Cd * clear_opening[x-1] * (DCp)**0.5 * wind_vel * 3600)
      else:
        clear_opening.append(0)
```

```
            vol_flow_thermal.append(0)
            vol_flow_wind.append(0)
        else:
            clear_opening.append(vent_data.loc[0, f'win{x}_width'].sum() * vent_data.loc[0, f'win{x}_height'].sum())
            vol_flow_thermal.append(1 / 3 * Cd * clear_opening[x-1] * (g * vent_data.loc[0, f'win{x}_height'].sum() * temp_diff /
avg_temp) ** 0.5 * 3600)
            vol_flow_wind.append(dim_less * clear_opening[x-1] * wind_vel * 3600)
        total_vol_flow.append((vol_flow_thermal[x-1] ** 2 + vol_flow_wind[x-1] ** 2) ** 0.5)

    resultant_air_change = max(total_vol_flow[0] + total_vol_flow[1],
                               total_vol_flow[2]) / (vent_data.loc[0, 'room_height'].sum() * tfa) * vent_data.loc[0, 'win_securrity'].sum() / 24
```

Go back

Function calculate_u_value

The assembly U-value is calculated in calculate_u_value. This function inputs the climate_type, new_assembly_data and new_assembly_lambda data and returns a new_assembly_Uval. This is hard coded for three sections each with one percentage (as is PHPPv9.6a).

The external and internal surface thermal resistances are automatically applied as follows:

```
intRsi_values = {'heating_predominant': {
                    'Roof': 0.1, 'Wall': 0.13, 'Floor': 0.17},
                    'temperate': {'Roof': 0.13, 'Wall': 0.13, 'Floor': 0.13},
                    'cooling_predominant': {'Roof': 0.17, 'Wall': 0.13, 'Floor': 0.1}}
                extRsi_values = {'Outdoor air': 0.04, 'Ground': 0}
```

Go back

Function win_U_value

As shown in the code snippet below this calculation is executed by the win_U_value function in the Windows.py module which takes as input the glazing type and frame type, the height and width of the window and how many sides of the window (sides_adjacent) have internal mullions. The function returns the U value as per ISO10077-1 including the window installation PSI value, the U value with no install psi value (as per Building Code) and the percentage of the window that is glazed (glz_per).')

```
FROM win_U_value(glazing_type, frame_type, width, height, sides_adjacent)
a_window = width * height
a_glazing = max(0,((width - (frame_type.frame_width * 2)) * (height - (frame_type.frame_width * 2))))
glz_per = (a_glazing / a_window) * 100
l_window = (width + height) * 2 * (1 - sides_adjacent/4)
l_glazing = (width - (frame_type.frame_width * 2)) * 2 + (height - (frame_type.frame_width * 2)) * 2
```

```
u_value = (a_glazing * glazing_type.U_glazing
        + (a_window - a_glazing) * frame_type.U_frame
        + l_window * frame_type.frame_install_psi
        + l_glazing * frame_type.frame_psi) / a_window
u_value_no_install_psi = (a_glazing * glazing_type.U_glazing
        + (a_window - a_glazing) * frame_type.U_frame
        + l_glazing * frame_type.frame_psi) / a_window
```

Function win_calc

The window glazing area is set from the window area and glz_per from the win_U_value function and the window orientation and angle set from the areas they are installed in. The windows are then grouped by orientation or if angle to horizontal is less than 30 degrees into Horizontal windows. This puts all windows into one of these five groups. This function inputs the window and areas and returns the windows table modified.

Go back

Function occupancy (see main_calc.py)

This function inputs the CFA and returns the occupancy.

ECCHO includes a default occupancy based on Conditioned Floor Area (CFA). This is calculated as follows:

```
return 1 + 1.9 * (1 - math.exp(-0.00013 * (tfa - 7) ** 2)) + 0.001 * tfa
```

Future versions after TBD will use an alternate default occupancy.

Number of occupants = 0.0168 * CFA + 0.5788

Note that in custom mode any manual occupancy can be entered.

Go back

Function location

This function inputs the country and then reads in the climate data for that country from Climate_data.csv and returns a location_list containing location and climate_zone.

Function climate_data

This function pulls the climate data from the Climate_data.csv file and dwelling_data and then formations it into location_climate_data_df. Note that the Climate_data.csv file already has the data mirrored about the equator as it is in PHPPv9.6a. This includes altitude adjustment using

```
height_adj = ( 0.6 / 100 * (  dwelling_data.loc[0, "height_above_sea"] - location_climate_data.loc[0, "Sea level m"]))
```

Function project_climate_type

This function takes as input the project_climate_file and uses the monthly temperatures to return the climate type.

Go back

Orientation of building and site to climate

Function areas_calc1

This function adjusts the input areas orientation to allow for the user to rotate the building. By adding the user-entered orientation to the areas orientation and then correcting to between 0 and 360 degrees, the building is rotated. Then the areas are mirrored about the equator for all dwellings with a latitude less than zero:

```
if float(latitude) < 0:
    areas["dev_nth"] = areas["dev_nth"].apply( lambda x: 180 - x if 180 - x >= 0 else 180 - x + 360 )
```

Note this is not the same as simply adding 180 degrees as the building is mirrored about the equator so walls facing east stay east and so on.

Go back

Function hot_water_calc

This function inputs shower_df, location_climate_data_df, dwelling_data, appliance_dhw and returns hot_water_data including the appliance_dhw per sqm of CFA (or TFA).

The annual kWh for generation of hot water is then calculated from the daily demand and the temperature difference:

```
hot_water_data['daily demand'] = hot_water_data['Time of use per use'] \
            * hot_water_data['Flow rate'] *  hot_water_data['Amount of uses according to type of use'] *
hot_water_data['occupancy']
  hot_water_data['annual kWh'] = hot_water_data['daily demand'] * (hot_water_data['Useful temperature'] -
avg_cold_water_temp) * 365 * 4.19 / 3600
```

Finally, the appliance_dhw is added and the sum divided by the reference area CFA (TFA in the code). Note appliance_dhw is calculated in the custom_appliances function.

Function hot_water_storage

This function calculates the storage and pipe losses for DHW. This function inputs the location_climate_data_df, dwelling_data pulling in the 'cylinder_size', fitting_loss inputs and cylinder location and returns the storage and pipe losses per reference area. Note that currently the storage cylinder is hard coded to 60C, hot water flow temperatures to 55C, external pipe diameter to 0.015m and 6 tap openings per day.

Pipe losses (W/K) are based on an assumed pipe length, which is calculated as follows:

pipe_length (m) = Conditioned Floor Area^(1/3) * tapping_points + 10

...where tapping points = 1+CFA/50.

An additional loss is then included for the fittings and 2m of pipework off the top of any cylinder (again, if present). This is based on the extent to which these are insulated.

```
fitting_loss = {'-':0,'Uninsulated':1,'Normal':0.5,'Medium':0.25,'Excellent':0.1}
    standby_loss_rate = heat_loss_rate * (CYLINDER_TEMP - cylinder_ext_temp)
    standby_losses = standby_loss_rate * 8.76 * dwelling_data.loc[0, 'cylinder_qty']

    tapping_points = 1+round(dwelling_data.loc[0, 'building_TFA']/50,0)

    pipe_length = dwelling_data.loc[0, 'building_TFA'] ** (1/3) * tapping_points + 10
    pipe_length_per = pipe_length / tapping_points
    water_vol = math.pi * ((EXT_PIPE_DIA - 0.0045)**2) / 4 * pipe_length_per
    pipe_vol = math.pi * (EXT_PIPE_DIA**2) / 4 * pipe_length_per - water_vol
    heat_loss_per = (1.16 * water_vol + 0.555 * pipe_vol) * (DHW_FLOW_TEMP - dwelling_data.loc[0, 'winter_int_temp'])
    tap_open_per_year_person = TAP_OPENINGS * 365
    pipe_loss = tap_open_per_year_person * heat_loss_per * dwelling_data.loc[0, 'occupancy']
```

Go back

Function custom_appliances

This function inputs appliances, dwelling_data as well as reading in appliance_data.csv, appliance_star_rating_data.csv. This function then returns the appliance_elec_demand, appliance_dhw, appliances. Note the appliance_elec_demand and appliance_dhw are annual kWh/m2.

The total hot water energy demand from dishwashers and washing machines (if connected to the domestic hot water supply) is calculated by setting the electricity fraction and then calculating per the code snippet below. If the DHW connection is set to yes for dishwasher,

the 'electricity fraction' is set to 0.5, for washing machine to 0.55 and for a washer-dryer combination machine to 0.8.

```
appliances['non elec fraction'] = 1 - appliances['electricity fraction']
appliances['appliance_non_elec'] = appliances['in_use'] * appliances['frequency'] * appliances['ref quantity'] *
appliances['non elec fraction'] * appliances['Norm demand'] / tfa
appliance_dhw = appliances.loc[appliances['appliance_type'] != 'Cooker',['appliance_non_elec']].values.sum()
```

Appliance electrical demand is calculated from:

```
appliances['appliance_elec_demand'] = appliances['in_use'] * appliances['frequency'] * appliances['ref quantity'] *
appliances['electricity fraction'] * appliances['Norm demand'] / tfa
appliance_elec_demand = appliances.loc[appliances['appliance_type'] != 'Cooker',
['appliance_elec_demand']].values.sum() + (100 * occupancy) / tfa
```

Note that the cooktop is the only appliance that allows for non-electricity.

```
cooker_norm_demand = {'Electricity':0.20,'LPG':0.25,'Natural Gas':0.25}
```

Also note that the clothes dryer has the utilisation factor set to 0.88 hard coded in this function to account for some clothes line usage.

Go back

Function lighting_calc

This function inputs the lighting_df and then returns the gross_lighting_demand / CFA

The lighting_watts is calculated for each area group from the quantity of lights and the lamp_watts. Then the gross_lighting_demand from:

```
gross_lighting_demand = 365 * (lighting_watts_total['Living areas'] * 2.2 + lighting_watts_total['Bedrooms'] * 1.1
                        + lighting_watts_total['Other areas'] * 0.7) / 1000
gross_lighting_demand / self.dwelling_data.loc[0, 'building_TFA']
```

Go back

Function aux_calc

This function inputs location_climate_data_df, vent_data, vent_calc, mvhr_units and then returns the aux_demand/CFA.

```
elec_efficiency = mvhr_units.loc[mvhr_units['description'] == vent_data.loc[0, 'mvhr'], 'elec_efficiency'].values.sum()
winter_fan_demand = 0.35 * (location_climate_data_df.loc[1:12, 'heating_days'].values.sum() * 24/1000) * elec_efficiency *
vent_calc['Air volume']
summer_fan_demand = vent_data.loc[0, 'vent_rate'] * (8.76 -
(location_climate_data_df.loc[1:12,'heating_days'].values.sum() * 24/1000)) * elec_efficiency * vent_calc['Air volume']
boiler_pump_demand = 0 # need to code this, check if central heating and add
aux_demand = winter_fan_demand + summer_fan_demand + boiler_pump_demand
```

```
aux_demand / self.dwelling_data.loc[0, 'building_TFA']
```

Go back

Function final_energy see final_energy_carbon.py

This function sort all the energy data into end_uses for tabular display and graphing. This includes adding the refrigerant leakage carbon impact into the total CO2 emissions.

Function set_emissions see models.py class Refrigerant

This reads in the refrig_properties.csv and calculates the co2_emissions from:

```
refrig_data = pd.read_csv("static/refrig_properties.csv")
    refrig_gwp = refrig_data.loc[int(self.refrig_type),'GWP']
    self.co2_emissions = ((self.leakage_rate/100 * self.system_life + self.eol_loss_rate/100)
                * refrig_gwp * self.refrig_charge) / self.system_life
```

Go back