GridLAB-D Course Tutorial 3

Load modeling

David P. Chassin

Summer 2016

dchassin@stanford.edu

Outline of Tutorial

Collectors

- Aggregating properties over groups of objects

Schedules and loadshapes

- Driving properties using schedules
- Load shape generation

Weather

- Driving models using weather data

Residential loads

- General house model with end-use loads and appliances

Collectors

Objects that record aggregate data from groups of objects.

Output object

Implemented in tape and mysql modules

- One of two primary ways of collecting data
- Observes aggregates of properties over a group of objects

Several possible destinations for tape data

- File : destination is a specially formatted files
- ODBC : destination is an ODBC database
- Memory : destination is a global variable

Group

Defines group criteria for objects

- Basis over which aggregation is performed
- Search only on init
- Initial search result is reused after first search
- → Groups are constant over time

Search criteria use invariant properties of objects

- Header properties: class, parent, rank, latitude, longitude, in_svc, out_svc, name, groupid
- Example: group "class=house";

Can include multiple groupings

- Example: group "class=house AND groupid=feeder1"

Property

Same as recorder, but with aggregators

- count, min, max, avg, std, sum, prod, mean, var, kur, gamma

Parts used for complex values

- real, imag, mag, ang, arg

Example: property "sum(power.mag)"

File

Similar rules to recorder object

- Meaning differs based on filetype property
- Format can be system specific
  - *"/" are normally used; "\" allowed in Windows*
- File must be writeable
- Path to file is not automatically created
- Existing file are overwritten
- Write failure is warned, but ignored

Interval

Determine the sampling interval for data

- How often is aggregate recomputed?

Units are seconds

- -1 means sample when aggregate changes
- 0 means sample each iteration

Limit

The maximum number of samples

- How many times will I sample?

Limits the size of the output file

- 0 is default
- 0 means no limit
- Similar to recorders, this will “drive” the simulation

Trigger

Specifies condition to start recording

- Works only for first target property
- Usual compare ops apply

Examples

- “< 0” : starts recording when aggregate is negative

Once triggered, recording continues to limit

Example

```
object collector {
        group "class=house AND groupid=feeder1";
        property "sum(power.mag),avg(hvac_load.real)";
        interval 3600;
        limit 24;
}
```

Demo

```
// demo_2_4: collector for sum and average of real part of all lights in ten houses
module residential{
          implicit_enduses NONE;
}
module tape;
clock {
          timezone PST+8PDT;
          starttime '2001-01-01 0:00:00 PST';
          stoptime '2001-07-01 00:00:00 PST';
}
schedule light_demand {
          * 1-3 * * * 0;
          * 4-6 * * * 0.15;
          * 7-19 * * * 0;
          * 20-0 * * * .85;
}
object house:..10 {
          cooling_setpoint 90 degF;
          object lights {
                    shape "type: analog; schedule: light_demand; power: 1.1 kW";
          };
}
object collector {
          file theat_collector.csv;
          group “class=lights”;
          property sum(energy.real),avg(energy.real); // no spaces, use real part only
          interval 3600;
          limit 744;
}
```

Schedules and Loadshapes

Using loadshapes and schedules to drive models

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Schedule Object

Implemented in gldcore (not an actual object)

- One of three primary ways of inputting dynamic data (others are players and shapers)

Represent a recurring pattern of values over time

- Minutes, hours, days, months, weekdays
- Define when a value is used:

```
minutes hours days months weekdays value
```

- Sunday is both day 0 (actual Sunday) and day 7 (holiday)

Schedules values and blocks

Format of time specification: POSIX standard (similar to *crontab* format in UNIX)

* 12 * * * X # value is X every day from noon to 1:00 PM

* 12-15 * 1-4 * Y # value Y from noon to 3pm Jan thru Apr

* 15,18-5 * * 1-5 # M-F 3 to 4 pm and 6 pm to 5 am

Schedule blocks

- Schedules blocks basis for normalization
- Schedules blocks must be full (no gaps in time)
- Size limitation of 63 nonzero unique values per block
- Max number of 4 blocks

Schedule options

Behavior can be modified using block options

- Normalization: values are rescaled within each block
  - *"normal" : normalize by sum of unweighted values in block*
  - *"absolute" : normalize by sum of absolute values in block*
  - *"weighted" : normalize by sum of time-weight values in block*
- "non-zero" : ensure no zero values are present in block
- "positive" : ensure no negative (or zero) values are present
- "boolean" : ensure values are strictly Boolean
- "interpolate" : values are interpolated over time

```
schedule demand {
    weighted;
    * 21-8 * * 1-5 1.2 # weekdays 9pm-9am, weeknights
    * 9-20 * * 1-5 1.5 # weekdays 9am-9pm, weekdays
    * * * * 6-0    0.8 # weekends, holidays
 }
```

Example Schedule

```
schedule heating_schedule {
       // winter weekdays
       *   0-5 * 1-4,10-12 1-5 65;     // sleep
       *   6-8 * 1-4,10-12 1-5 70;     // awake
       *  9-17 * 1-4,10-12 1-5 60;     // away
       * 18-20 * 1-4,10-12 1-5 70;     // awake
       * 21-23 * 1-4,10-12 1-5 65;     // sleep

       // winter weekends
       * 0-9;21-23 * 1-4,10-12 6-7 65; // sleep
       *     10-20 * 1-4,10-12 6-7 70; // awake

       // summer
       *           *   * 5-9     * 60; // all summer
}
```

Schedule Implementation

There are two ways to use schedules

- **Schedule transformation**
  - Available methods to define transformation
    - *Linear mapping function (internal function)*
    - *Non-linear mapping function (external function)*
    - *Discrete-time transfer function (z-domain filter)*
  - Drives property from the current value of the schedule
- **Loadshapes**
  - Schedule alters energy or power of loadshape property

Schedule Transforms

Works like targeting a player to an object property

```
object house {
            heating_setpoint heating_schedule*2+3;
}
```

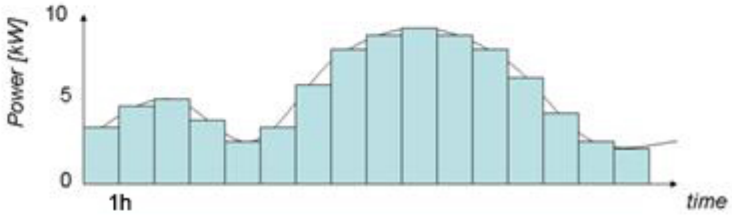
Syntax and order are important!

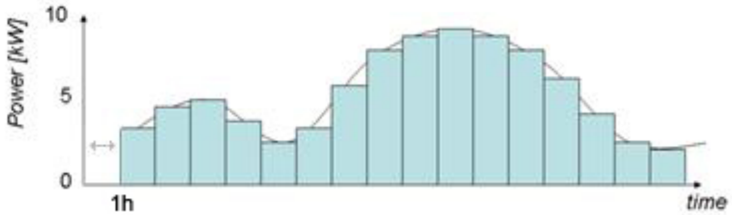
- Very limited valid linear operators
  - *Only multiplication, addition, and subtraction.*

Schedule skew

One schedule can be skewed in time

- Time skew differs for each object
- Units of skew are in second
- No unit conversion






```
object house {
     schedule_skew 3600;
     heating_setpoint heating_schedule;
}
```

Schedule caveats

Differences from the cron syntax:

- Alternate day and weekday not supported
  - *When both day and weekday given it is not "day or weekday"*
- Step by syntax (using /) is not supported.
- Special keywords (e.g., @hourly, @daily) not supported
- Weekday 7 refers to holidays the occur on weekday
  - *Holidays are not supported yet, but will be someday*

All times are considered in local time

- Scheduled changes during daylight-savings/summer time (DST) shifts could result in a missing or duplicate value.

Load shapes

Property – loadshape

- Special property type (pseudo-double)
- Driven schedules

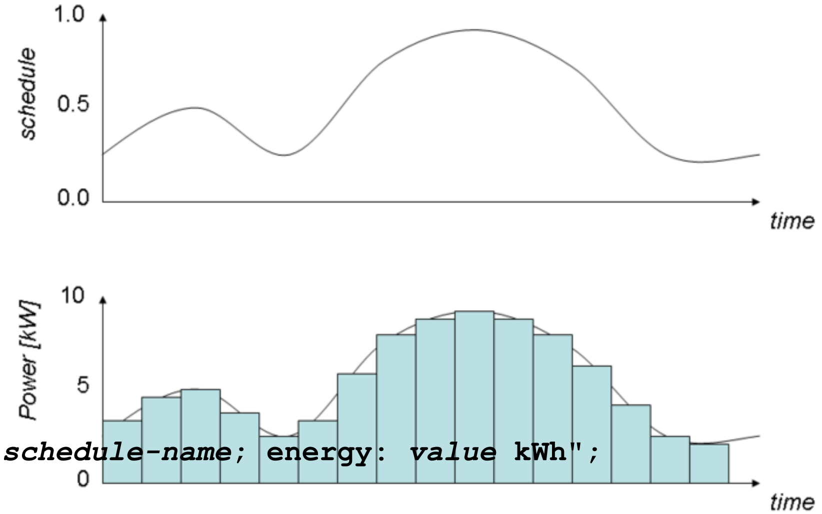
5 types

- Analog
- Pulsed
- Modulated (un-validated)
- Queued
- Scheduled

Analog Loadshapes

Directly compute power from values in the schedule

- An analog loadshape is defined using one of the three following methods

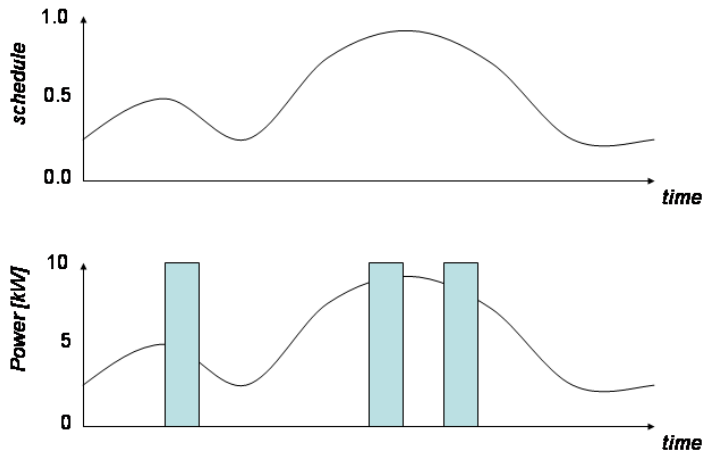



```
class example {
        loadshape myshape;
}
// Fixed scheduled energy in a block
object fixed-energy {
        myshape "type: analog; schedule: schedule-name; energy: value kWh";
}
// Fixed power ( schedule value times the power value )
object scaled-power {
        myshape "type: analog; schedule: schedule-name; power: value kW";
}
// Unscaled ( value in schedule is used directly )
object unscaled {
        myshape "type: analog; schedule: schedule-name";
}
```

Pulsed

Emits pulses at random times

- Total energy is define over the period of the loadshape.
- A pulsed loadshape is defined using one of the two methods:

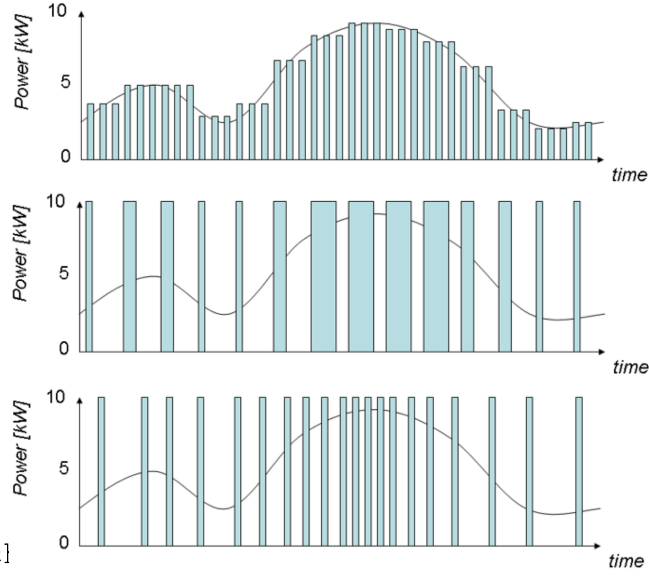



```
class example {
  loadshape myshape;
}
// Constant pulse duration
object pulse-width {
  myshape "type:pulsed; schedule:name; energy:value kWh; count:value; duration:value s";
}
// Constant pulse power
object pulse-amplitude {
  myshape "type:pulsed; schedule:name; energy:value kWh; count:value; power:value kW";
}
```

Modulated Loadshapes

Emit modulated pulses, 3 alternatives

1. constant period and duty-cycle (amplitude)
2. constant power and on-time (pulsewidth)
3. constant power and period (frequency).



Example:

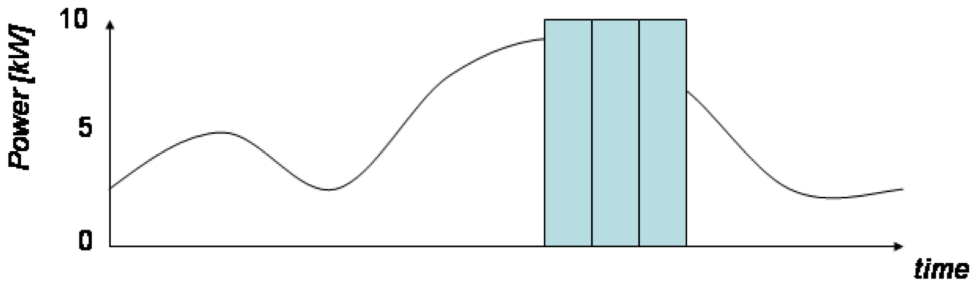
```
class example {
        loadshape name;
}
object sample {
        myshape "type: modulated; modulation: amplitude; sc
kWh; count: number; period: time s";
}
object sample {
        myshape "type: modulated; modulation: pulsewidth; schedule: schedule-name;
energy: value kWh; count: value; power: value kW";
}
object sample {
        myshape "type: modulated; modulation: frequency; schedule: schedule-name;
energy: value kWh; count: value; power: value kW";
}
```

Queued Loadshapes

Random pulses when a queue

- Accrues from the loadshape to a "on" threshold
- Emitting pulses until queue reaches "off" threshold.

Examples:




```
class example {
        loadshape myshape;
}
// Constant pulse duration
object sample {
        myshape "type: queued; modulation: pulsed; schedule: schedule-name; energy: value kWh; count: value; duration: value s; q_on: value; q_off: value";
}
// Constant pulse power
object sample {
        myshape "type: queued; modulation: pulsed; schedule: schedule-name; energy: value kWh; count: value; power: value kW; q_on: value; q_off: value";
}
```

Demo

```
// This demo uses a schedule for
// a plugload object and records the power demand and energy used
// by the object.
module tape;
module residential{
          implicit_enduses NONE;
}
clock {
          timezone PST+8PDT;
          starttime '2001-01-01 00:00:00';
          stoptime '2001-01-02 00:00:00';
}
schedule plug_shape {
          * 0-2 * * * 0.15;
          * 3-5 * * * 0.35;
          * 6-8 * * * 0.70;
          * 9-17 * * * 0.500;
          * 18-20 * * * 0.86;
          * 21-23 * * * 0.25;
}
```

Demo (cont.)

```
object house {
        object plugload {
           shape "type: analog; schedule: plug_shape; power: 1kW";
           object recorder {
              property power,demand,energy;
              interval 3600;
              limit 24;
              file plugload_energy_usage.csv;
           };
        // Required to implement electrical properties - more detail later
           power_factor 1.0;
           power_fraction 1.0;
           current_fraction 0.0;
           impedance_fraction 0.0;
        };
};
```

Questions?

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Climate

Using weather data to drive models

Purpose of climate data

Many systems affected by weather

- Loads, generators, lines, etc.

Several sources of weather data

- Typical meteorological year (TMY)
- NOAA weather records
- Other weather station records

Be aware of issues with each source

TMY Data

Only source supplied with GridLAB-D

- Some issues with long-term simulations
  - *Months are seamed with discontinuities*
  - *Represent 12 different typical months, not a typical year*
- Data files are large
- Default data type for climate files
- Latitude, longitude, city, state loaded at init
- Temperature, humidity, solar, and wind updated hourly for TMY2 data
  - *Quadratic and Linear interpolations are supported*
- Does not drive simulation

Downloading TMY data

Repository at SourceForge

- $SVN/code/data/
- Only US cities are supported now
- Files are zipped by state

Download process

- Locate state zip file(s) in repository
- Extract desired *.tmy2 file into working directory
- Ok to extract entire ZIP for state in shared/gridlabd folder

CSV Reader

Allows user to import own weather data

Supports all variables currently used

- Variables to be used are specified on first line:
  - *temperature, wind_speed, etc.*
- First column is always the timestamp

Some syntax issues

- Uses # for comments
- Uses $ for state and city names and latitude and longitude specifications
- No unit conversions
- Does not support interpolations

Example CSV Reader

```
#sample weather CSV file
# timestamp format is "month:day:hour:minute:second"
#   reads left to right - m:d:h is assumed if only 3 values
$state_name=California
$city_name=Berkeley

temperature,humidity
01:01:04, 50, 0.05 // Jan 1 4am temp=50, humidity=0.05
01:01:08, 62, 0.16 // Jan 1 8am temp=62, humidity=0.16
01:01:13, 78, 0.12 // Jan 1 1pm temp=78, humidity=0.12
01:01:20, 70, 0.10 // Jan 1 8pm temp=70, humidity=0.10
01:02:02, 51, 0.06 // Jan 2 2am temp=51, humidity=0.06
```

Climate class parameters

```
class climate {
        char32 city;
        char1024 tmyfile;
        double temperature[degF];
        double humidity[%];
        double solar_flux[W/sf];      // 9 orientations
        double wind_speed[mph];
        double wind_dir[deg];
        double wind_gust[mph];
        double record.low[degF];
        double record.high[degF];
        double record.solar[W/sf];
        enum interpolate; //NONE (default), LINEAR, QUADRATIC
}
```

Examples

```
// Example 1: Using tmy2 data
object climate {
        tmyfile "name_of_file.tmy2";
}
// Example 2: Using a csv file
object csv_reader {
        name "reader-name";
        filename "name_of_file.csv";
}
object climate {
        tmyfile "name_of_file.csv";
        reader "reader-name";
}
```

Questions?

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Residential buildings

Modeling residential building loads

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Two kinds of models

House E (ETP)

- Class name is "house"
- Based on equivalent thermal parameters (ETP) model
- Second-order model (include single thermal mass effects)

Appliance models included in module

- Built-in appliance (explicit) or load shapes/enduses (implicit)
- Explicit models
  - *Some use performance models*
  - *Others use physical/state-space models*

Note: House A (ASHRAE) is deprecated but still in code

House Object

Has two main functions

- Determines state and power consumption of the HVAC system (ETP Model)
- Accumulates effects of residential appliances

Attaches to the power system via a triplex_meter

- Triplex_meter must be parent of house
- Absent the meter, voltage source is static 120/240 V

Uses climate object for weather data

Physical vs. Implicit Models

SLAC

Only HVAC and waterheater currently incorporate physical models

- Use physical parameters to compute state
- Mainly used for thermostatic control studies
- Dryer, dishwasher, electric range, clothes washer, and refrigerator models are fully developed performance models
- Freezer and microwave are experimental models
- Evcharger is a probabilistic model

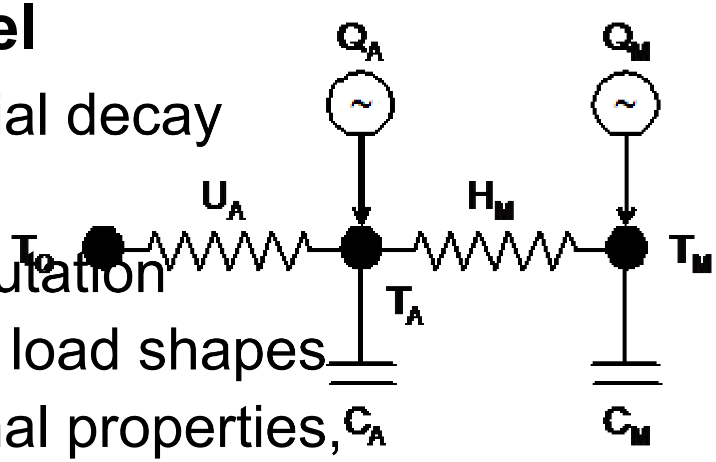
Not all appliances use residential enduse loadshapes

- Some use time varying ZIP models

Equivalent Thermal Parameters (ETP)

SLAC

Two-node lumped-parameter model



- Over-damped DC circuit – exponential decay
- Simple enough for direct analytic (not numerical) solution & fast computation
- Complex enough to capture building load shapes
- Accounts for weather, building thermal properties, solar & internal gains, thermostat settings
- Heat added by HVAC system + internal (appliances) + solar
- Internal gains driven by time-of-day, day-of-week schedule
- Solar gains from weather & window properties
- Split between air & mass nodes

HVAC capacity & COP depend on outdoor temperature

- QHVAC-electric = CapacityThermal / COP

Heat Balance Equations Solved

Heat balance for the air temperature node (TA):

$$0 = QA - UA(TA - TO) - HM(TA - TM) - CA\, dTA/dt$$

Heat balance for the mass temperature node (TM):

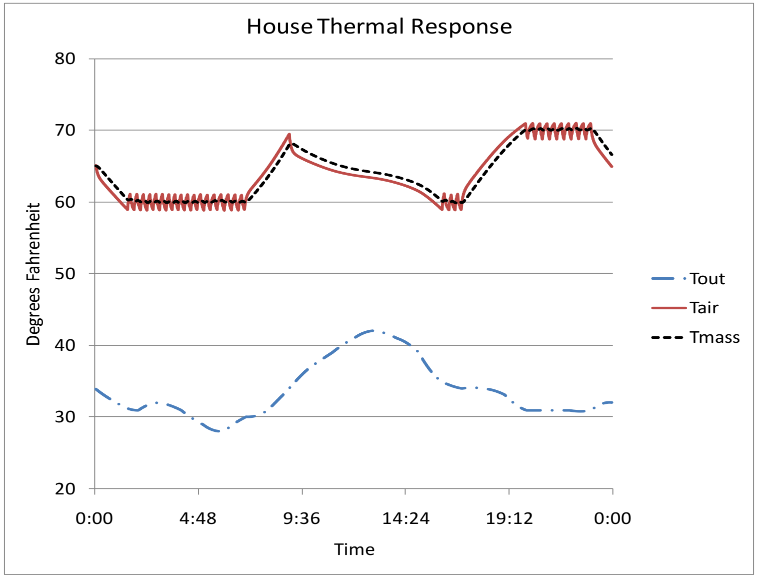
$$0 = QM - HM(TM - TA) - CM\, dTM/dt$$

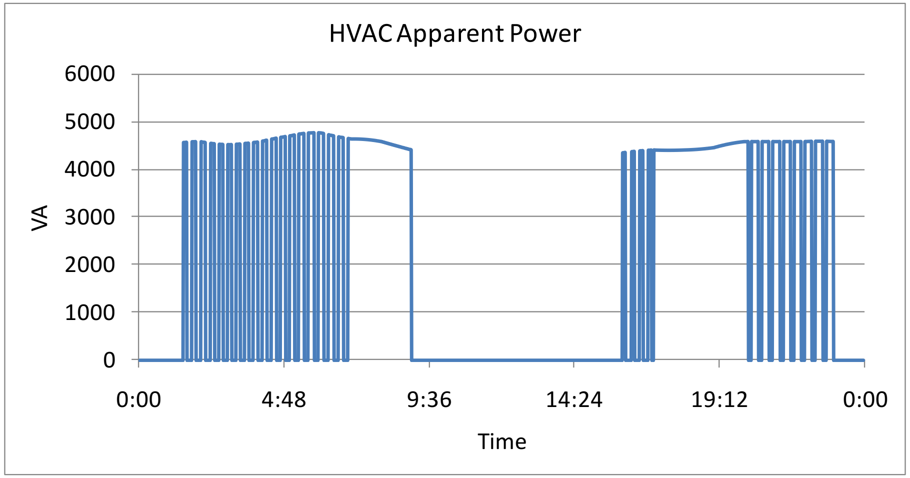
Solution is second-order differential equation

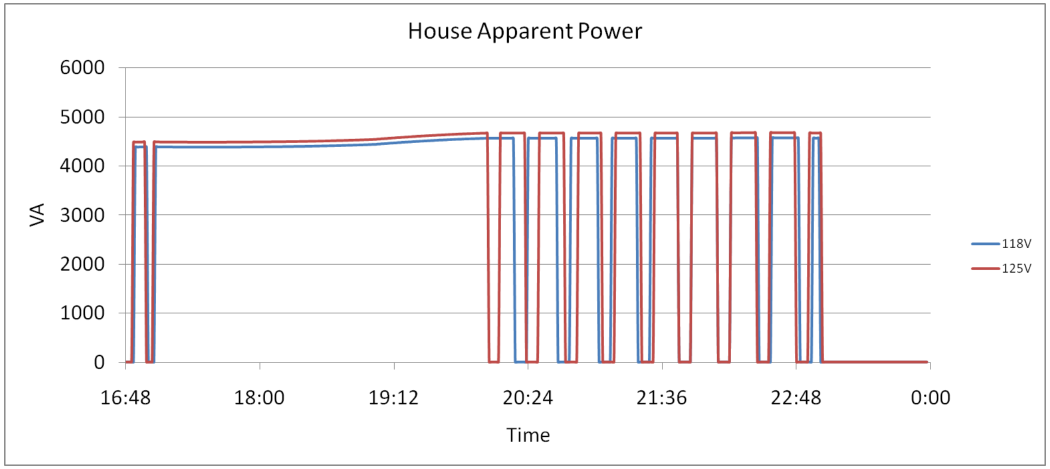
- Slow pole for longer mass effects (affects mid-term DR)
- Fast term for shorter mass effects (affects H/C cycling)

GridLAB-D solves ETP only when needed

Physical Load Model – Residential HVAC







House design parameters

```
object house {
    // Physical design parameters – all default to 'reasonable' values from floor area
    double floor_area[sf];
    double gross_wall_area[sf];
    double ceiling_height[ft];
    double aspect_ratio;
    double window_wall_ratio;
    double number_of_doors;
    double exterior_wall_fraction;
    double interior_exterior_wall_ratio;
    double exterior_ceiling_fraction;
    double exterior_floor_fraction;
    double number_of_stories;
    double Rroof[degF];
    double Rwall[degF];
    double Rfloor[degF];
    double Rwindows[degF];
    double Rdoors[degF];
    double window_shading;
    double window_exterior_transmission_coefficient;
    …
```

HVAC design parameters

```
object house {
        // HVAC design parameters – all default to 'reasonable' values from floor area
        //   Some values are extracted from climate data
        double cooling_design_temperature[degF];
        double heating_design_temperature[degF];
        double design_peak_solar[Btu/h];
        double design_internal_gains[W/sf];
        double cooling_supply_air_temp[degF];
        double heating_supply_air_temp[degF];
        double duct_pressure_drop[in];
        double heating_COP[pu];
        double cooling_COP[Btu/kWh];
        double design_heating_capacity[Btu/h];
        double design_cooling_capacity[Btu/h];
        double design_heating_setpoint[degF];
        double design_cooling_setpoint[degF];
        double auxiliary_heat_capacity[Btu/h];
        double over_sizing_factor[unit];
        …
```

Heatflow parameters

```
object house {
	// These values define the flow of heat from incident solar, air, and mass
	double solar_heatgain_factor;
	double airchange_per_hour;
	double internal_gain[Btu/h];
	double solar_gain[Btu/h];
	double incident_solar_radiation[Btu/h];
	double heat_cool_gain[Btu/h];
	double air_heat_fraction[pu];
	double mass_heat_capacity[Btu/degF];
	double mass_heat_coeff[Btu/degF];
	double air_heat_capacity[Btu/degF];
	double total_thermal_mass_per_floor_area[Btu/degF];
	double interior_surface_heat_transfer_coeff[Btu/h];
	double design_internal_gain_density[W/sf];
	…
```

Fan and Motor parameters

```
object house {
        // Fan design parameters default to reasonable values from HVAC designs
        double fan_design_power[W];
        double fan_low_power_fraction[pu];
        double fan_power[kW];
        double fan_design_airflow[cfm];
        double fan_impedance_fraction[pu];
        double fan_power_fraction[pu];
        double fan_current_fraction[pu];
        double fan_pow
        double hvac_motor_efficiency[unit];
        double hvac_motor_loss_power_factor[unit];er_factor[pu];
        …
```

Thermostat parameters

```
object house {
        // These parameters control the operation of the thermostat
        double heating_setpoint[degF];
        double cooling_setpoint[degF];
        double aux_heat_deadband[degF];
        double aux_heat_temperature_lockout[degF];
        double aux_heat_time_delay[s];
        double thermostat_deadband[degF];
        int16 thermostat_cycle_time;
        timestamp thermostat_last_cycle_time;
        int64 last_mode_timer;
        …
```

Derived parameters

```
object house {
	// These parameters are all derived from others, or calculated over time
	double air_temperature[degF];
	double outdoor_temperature[degF];
	double mass_temperature[degF];
	double air_volume[cf];
	double air_mass[lb];
	double latent_load_fraction[pu];
	double heating_demand;
	double cooling_demand;
	double envelope_UA[Btu/degF];
	double airchange_UA[Btu/degF];
	…
```

Load parameters

```
object house {
        // These parameters translate appliance operations into electrical demand
        enduse panel;
        complex panel.energy[kVAh];
        complex panel.power[kVA];
        complex panel.peak_demand[kVA];
        double panel.heatgain[Btu/h];
        double panel.heatgain_fraction[pu];
        double panel.current_fraction[pu];
        double panel.impedance_fraction[pu];
        double panel.power_fraction[pu];
        double panel.power_factor;
        complex panel.constant_power[kVA];
        complex panel.constant_current[kVA];
        complex panel.constant_admittance[kVA];
        double panel.voltage_factor[pu];
        double panel.breaker_amps[A];
        double hvac_breaker_rating[A];
        double hvac_power_factor[unit];
        double hvac_load;
        double total_load;
        …
```

Enumerations

```
object house {
    // Various enumerations that define different aspects of home, * is default value
    enum heating_system_type; //RESISTANCE, *HEAT_PUMP, GAS, NONE
    enum cooling_system_type; //HEAT_PUMP, ELECTRIC, *NONE
    enum auxiliary_system_type; //ELECTRIC, *NONE
    enum auxiliary_strategy; //LOCKOUT, TIMER, DEADBAND, NONE
    enum fan_type; //TWO_SPEED, *ONE_SPEED, NONE
    enum thermal_integrity_level; // VERY_GOOD, GOOD, ABOVE_NORMAL,
            // NORMAL, BELOW_NORMAL, LITTLE, VERY_LITTLE, *UNKNOWN
    enum glass_type; //*LOW_E_GLASS, GLASS, OTHER
    enum window_frame; //INSULATED, WOOD, *THERMAL_BREAK, ALUMINUM, NONE
    enum glazing_treatment; //HIGH_S, LOW_S, REFL, ABS, *CLEAR, OTHER
    enum glazing_layers; //THREE, *TWO, ONE, OTHER
    enum motor_model; //FULL, BASIC, *NONE
    enum motor_efficiency; //VERY_GOOD, GOOD, *AVERAGE, POOR, VERY_POOR
    …
```

“Easy” House – Major Parameters

object house {

}

```
object house {
   floor_area 2500;
}
```

```
object house {
   floor_area 2500;
   thermal_integrity GOOD;
   cooling_setpoint 72;
   heating_setpoint 65;
}
```

```
object house {
   floor_area 2500;
   Rwall 19;
   Rroof 30;
   Rfloor 22;
   Rdoors 5;
   Rwindows 2.1;
   airchange_per_hour 0.5;
   number_of_stories 1;
   cooling_setpoint 72;
   heating_setpoint 65;
   cooling_system_type ELECTRIC;
   heating_system_type HEAT_PUMP;
   auxiliary_system_type ELECTRIC;
}
```

Implicit Models
Residential Enduse

- **All residential objects can be used as a residential enduse**
- **Uses any of four available loadshapes (analog, pulsed, modulated, queued)**
- **Overrides physical state driven models, but certain variables are still used (e.g. *placement* still determines the where the heatgain goes)**
- **By default, all appliances are turned on as residential enduses**
  - Certain appliances may be turned on, or all of the turned off

```
module residential {
     implicit_enduses NONE;
     // or
     implicit_enduses LIGHTS|PLUGS;
}
```

Residential Enduse

```
object lights {
    // User assigned values. Lights are only used as an example.
    double power_factor[unit];
    double heatgain_fraction[pu];
    double installed_power[kW];
    double impedance_fraction[pu];
    double current_fraction[pu];
    double power_fraction[pu];
    loadshape shape;
    double breaker_amps[A];
    // Automatically updated values
    double heatgain[Btu/h];
    complex power[kVA];
    complex energy[kVAh];
    complex peak_demand[kVA];
    double voltage_factor[pu];
    double constant_current[kVA];
    double constant_power[kVA];
    double constant_impedance[kVA];
}
```

Residential Enduse (example)

```
class lights {
        loadshape myshape;
}
object lights {
        myshape "type: analog; schedule: schedule-name; energy: value kWh";
        installed_power 1 kW;
        power_factor 0.95;
        impedance_fraction 1.0;
        power_fraction 0.0;
        current_fraction 0.0;
        placement INDOOR;
}
```

Available Appliances

Waterheater

Lights

Refrigerator

Clotheswasher

Dishwasher

Plugs

Microwave

Range

Freezer

Dryer

Evcharger

Occupants

Waterheater (physical model)

The water heater simulation uses two very different models depending on the state of the tank at any given moment. They are:

<u>One-Node Model:</u>

- Simple, lumped-parameter: tank is at uniform temperature
- Computes time required to change temperature of entire mass

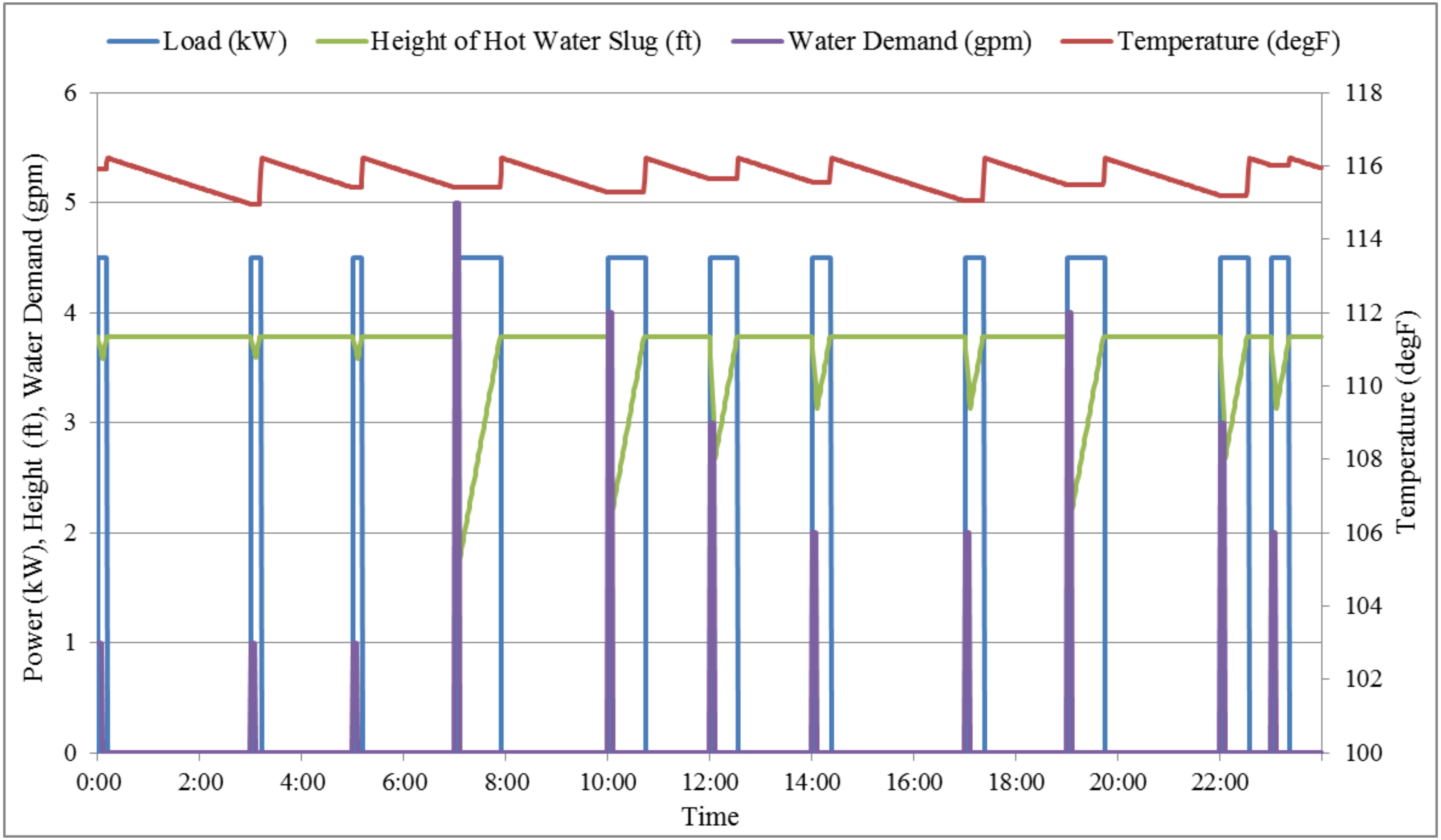
<u>Two-Node Model:</u>

- Two waster masses, each at different uniform temperature
- Upper “hot” node near the set point temperature
- Lower “cold” node is between inlet temperature and setpoint
- Computes motion of thermocline as heat is added/removed

Waterheater (physical model)

```
class waterheater {
        double tank_volume[gal];
        double tank_UA[Btu/h];
        double tank_diameter[ft];
        double water_demand[gpm];
        double heating_element_capacity[kW];
        double inlet_water_temperature[degF];
        enumeration heat_mode; //GASHEAT, ELECTRIC
        enumeration location; //GARAGE, INSIDE
        double tank_setpoint[degF];
        double thermostat_deadband[degF];
        double temperature[degF];
        double height[ft];
        double actual_load[kW];
}
```

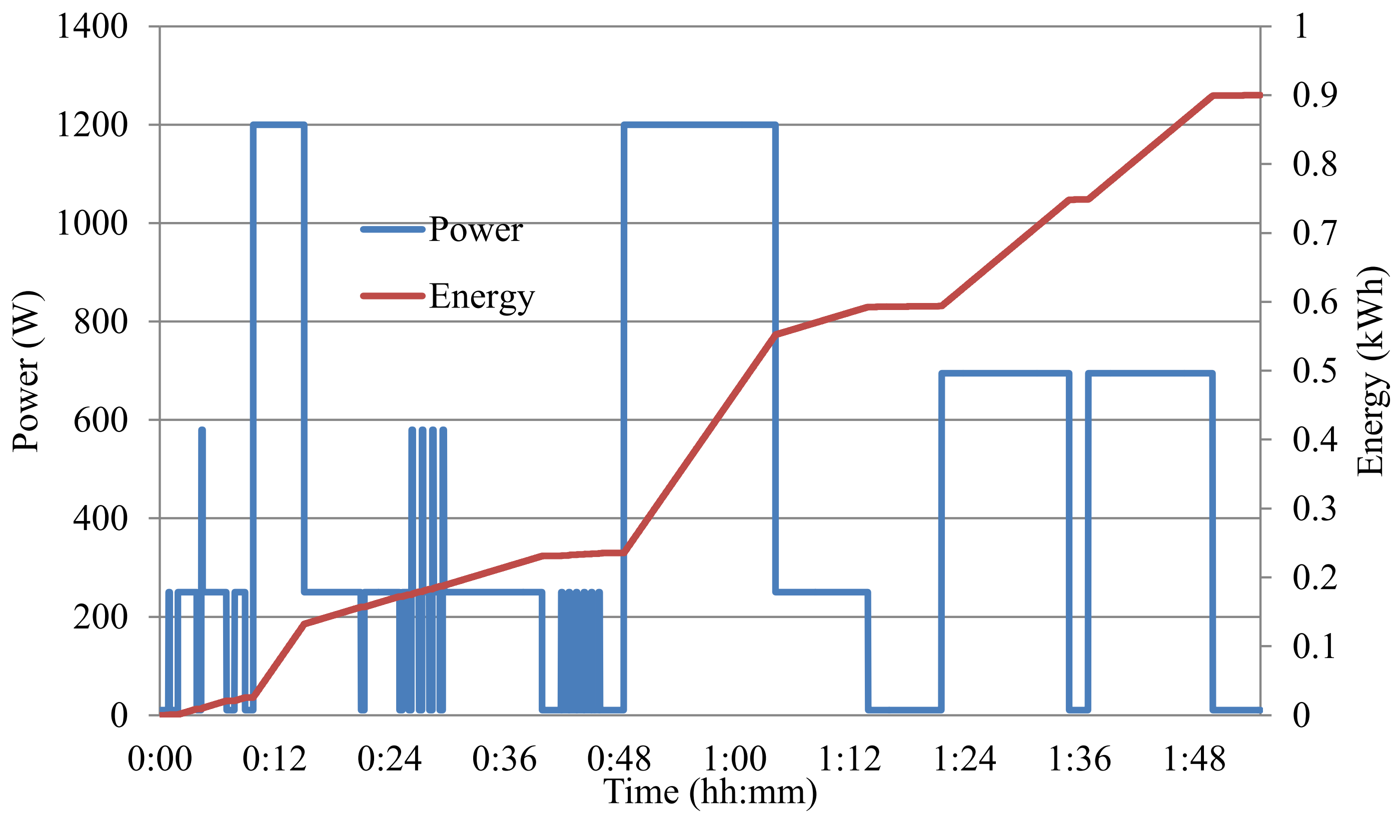
Waterheater (physical model)



Dishwasher

```
 object dishwasher {
   double energy_baseline [kWh];
   bool Heateddry_option_check;//// true or false
   double control_power [W];
   double motor_power [W];
   double dishwasher_coil_power_1 [W];
   double dishwasher_coil_power_2 [W];
   double dishwasher_coil_power_3 [W];
   double daily_dishwasher_demand DISHWASHER;
   double queue;
   double queue_min;
   double queue_max;
};
```

Dishwasher



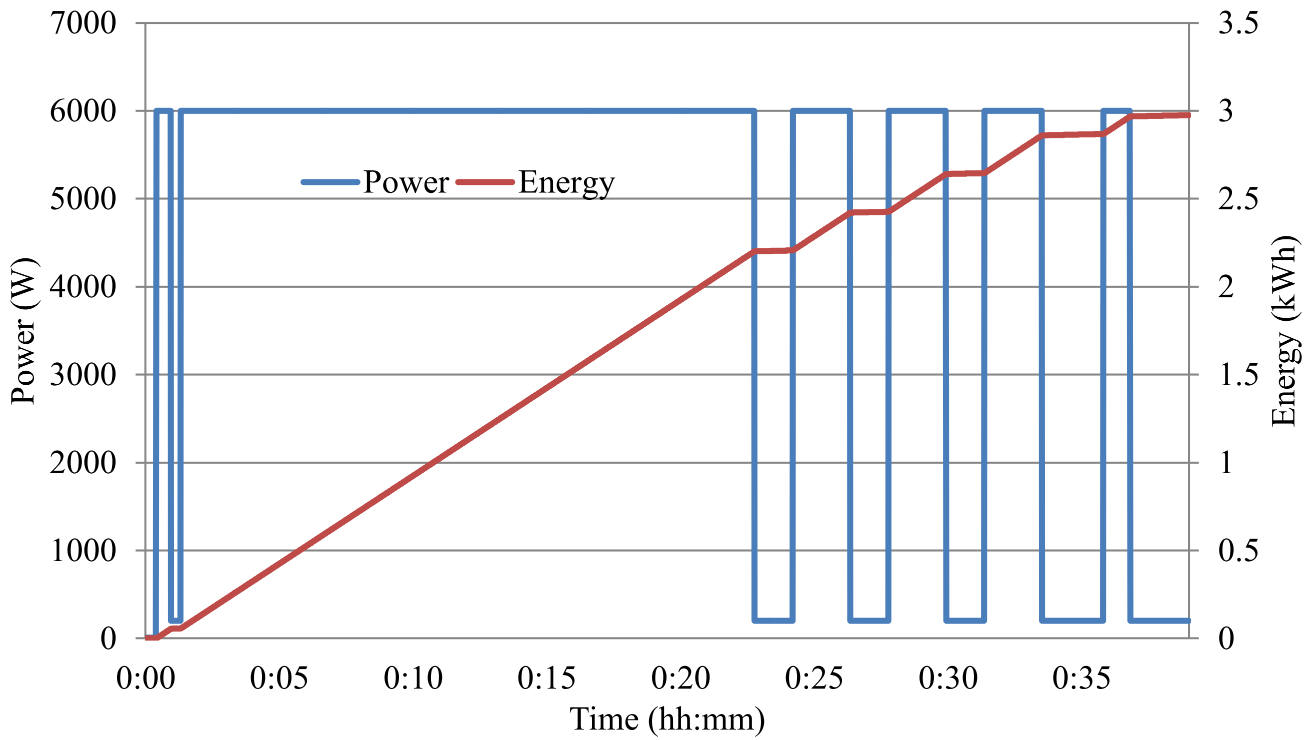
Variables	Values
energy_baseline	0.9kWh
Heateddry_option_check	true
control_power	10W
motor_power	250W
dishwasher_coil_power_1	580W

Variables	Values
dishwasher_coil_power_2	695W
dishwasher_coil_power_3	950W
queue	1
queue_min	0
queue_max	2

Dryer

```
object dryer {
    double energy_baseline [kWh];
    double controls_power [W];
    double motor_power [W];
    double dryer_coil_power [W];
    double daily_dryer_demand DRYER;
    double queue;
    double queue_min;
    double queue_max;
};
```

Dryer



Variables	Values
energy_baseline	3.0kWh
controls_power	10W
motor_power	200W
dryer_coil_power	5800W
queue	0.8
queue_min	0
queue_max	2

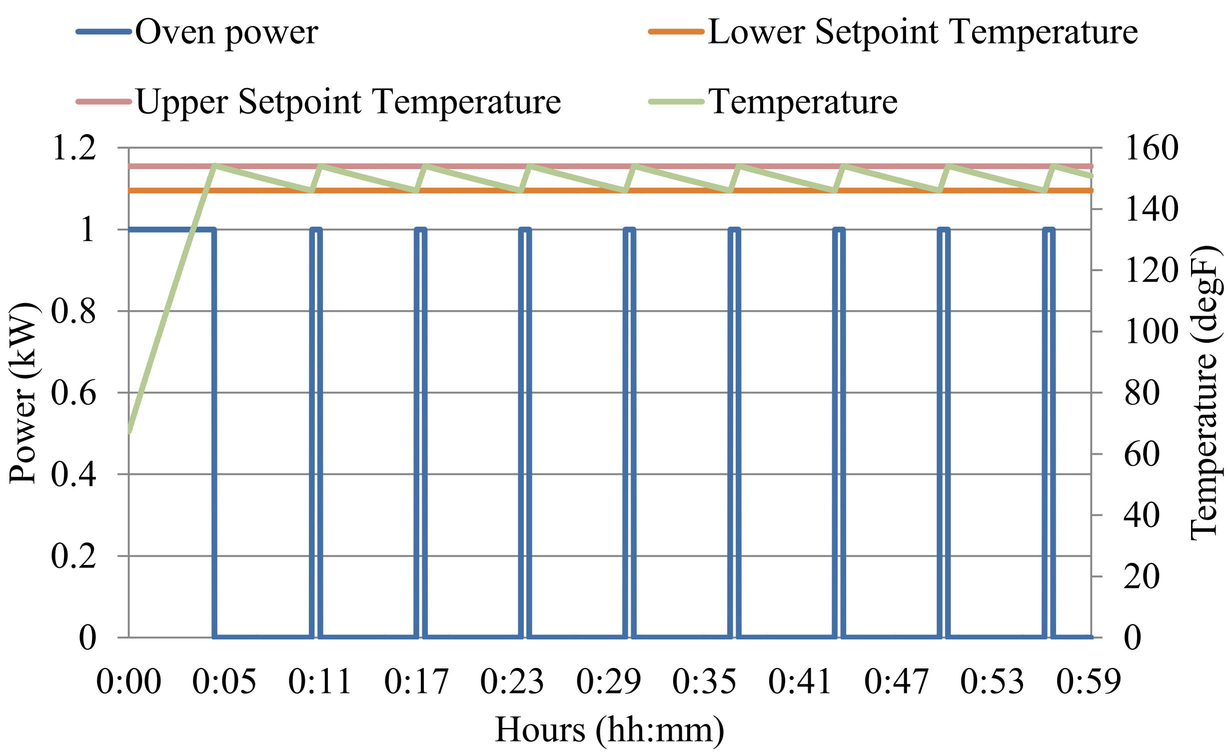
Range

```
object range {
   //Oven
  double oven_volume [gal];
  double heating_element_capacity [Kw];
  double oven_setpoint [degF];
  double temperature [degF];
  double thermostat_deadband [degF];
  enumeration location; //GARAGE, INSIDE
  double oven_UA [Btu.h/degF];
  double food_density [lb/cf] ;
  double specificheat_food [Btu/lb.degF];
  double time_oven_setting [s];
  double queue_oven;
  double demand_oven RANGE;
  double oven_demand;
```

Range

```
//cooktop
double cooktop_energy_baseline [kWh];
double cooktop_coil_setting_[kW];
double cooktop_coil_setting_2 [kW];
double cooktop_coil_setting_3 [kW];
double cooktop_interval_setting_1 [s];
double cooktop_interval_setting_2 [s];
double cooktop_interval_setting_3 [s];
double time_cooktop_setting [s];
double demand_cooktop RANGE;
double queue_cooktop;
double queue_min;
double queue_max;
};
```

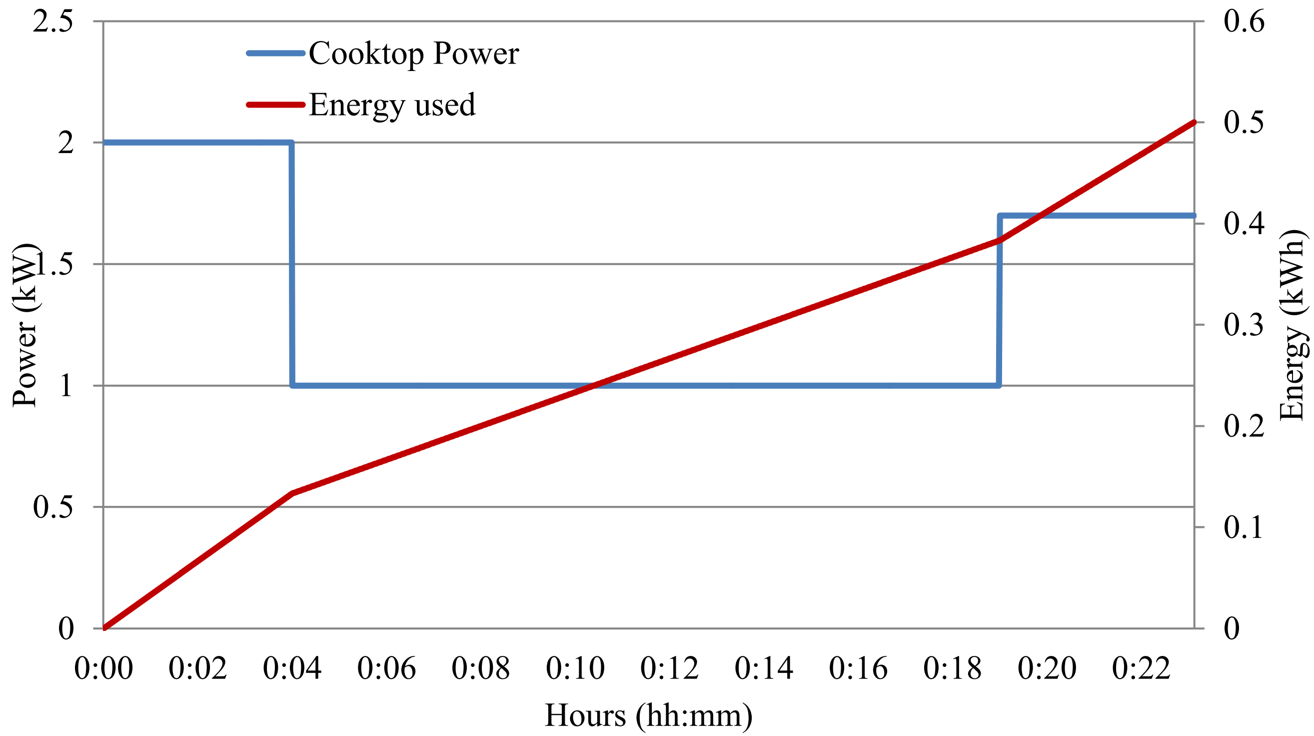
Range (Oven)



Variables	Values
oven_volume	5
heating_element_capacity	1kW
oven_setpoint	150degF
temperature	70degF
thermostat_deadban	8degF
location	INSIDE

Variables	Values
oven_UA	2.9
food_density	5
specificheat_food	1
time_oven_setting	3600s
queue_oven	0.85

Range (Cook top)



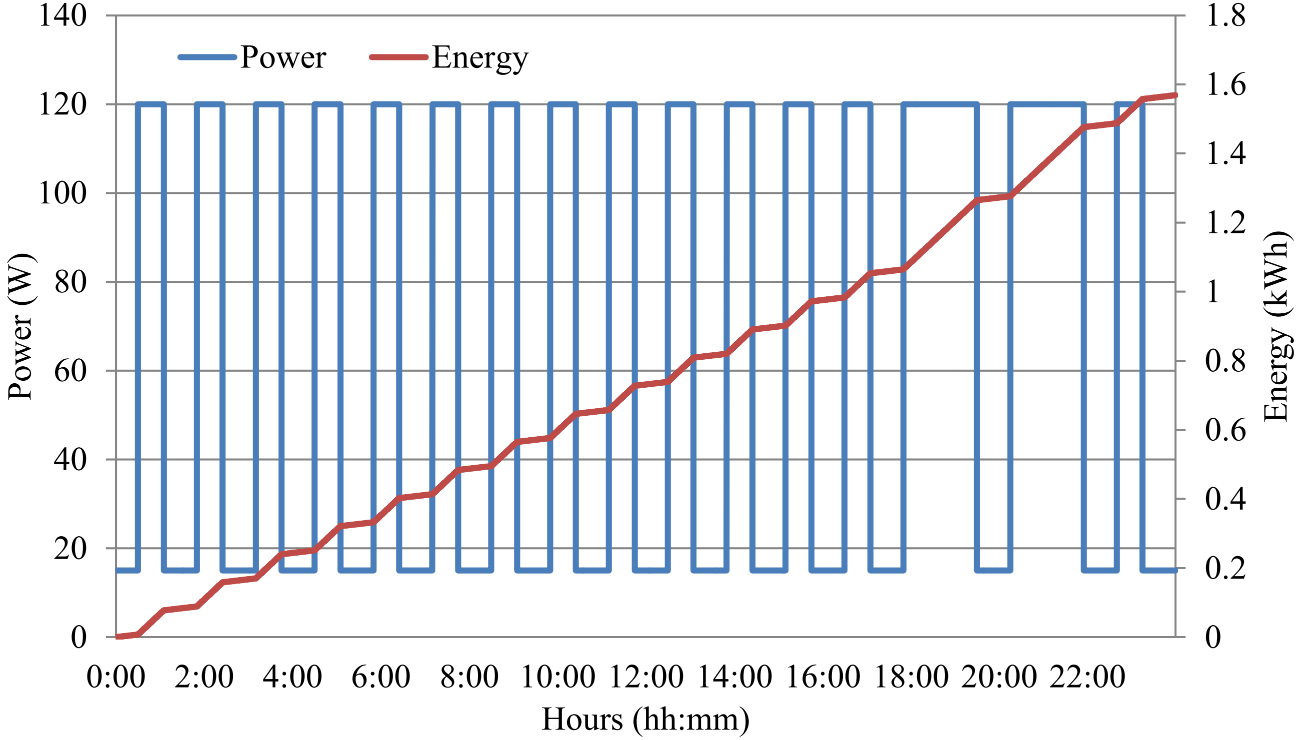
Variables	Values
cooktop_energy_baseline	0.5 kWh
cooktop_coil_setting_1	2kW
cooktop_coil_setting_2	1.0kW
cooktop_coil_setting_3	1.7kW
cooktop_interval_setting_1	240s

Variables	Values
cooktop_interval_setting_2	900s
cooktop_interval_setting_3	120s
Queue_cooktop	0.99
queue_min	0
queue_max	2

Refrigerator

```
object refrigerator {
  double door_opening_criterion REFRIGERATOR;// ELCAP
  double daily_door_opening;
  enumeration state; // COMPRESSSOR_OFF_NORMAL
  double energy_used [kWh];
  enumeration defrost_criterion;// DOOR_OPENINGS
  double delay_defrost_time [s];
  double door_opening_criterion;
};
```

Refrigerator

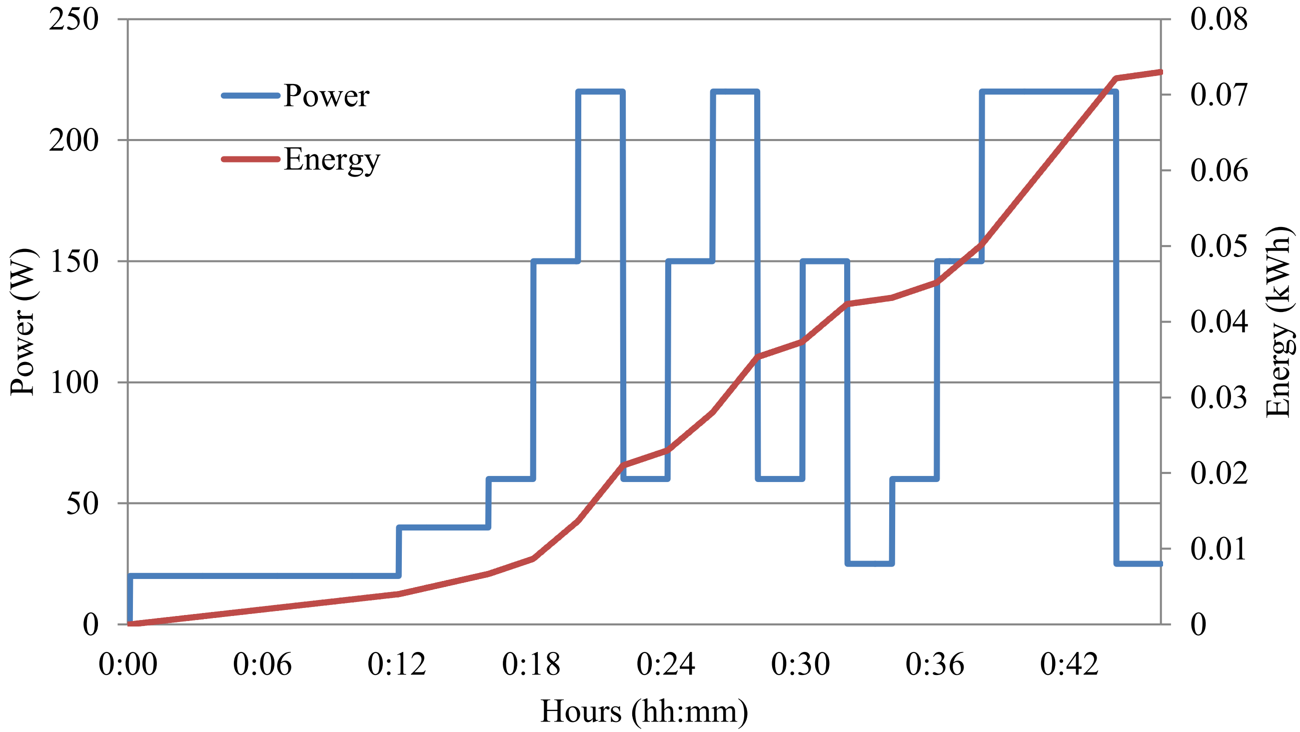


Variables	Values
door_opening_criterion	REFRIGERATOR
daily_door_opening	20
state	COMPRESSSOR_OFF_NORMAL
energy_used	13.5kWh
defrost_criterion	DOOR_OPENINGS
delay_defrost_time	600s
door_opening_criterion	24

Clotheswasher

```
object clotheswasher {
   double queue;
   double demand CLOTHESWASHER;//ELCAP
   enumeration state// STOPPED
   double queue_min;
   double queue_max;
};
```

Clothes washer



Variables	Values
queue	1.7
state	STOPPED
queue_min	0
queue_max	2

Lights

```
class lights {
  // In addition to residential enduse variables
  enumeration type; //HID, SSL, CFL, FLUORESCENT, INCANDESCENT
  enumeration placement; // OUTDOOR, INDOOR
  double installed_power[kW];
  double circuit_split;  // -1=100% on 1 … +1=100% on 2
  double demand[unit];   // fraction that is on
  double power_density[W/sf];
  complex actual_power[kVA]; //actual power consumption
}
```

Plugs

```
class plugload {
        // In addition to residential enduse variables
        double circuit_split;
        double demand[unit];
        double installed_power[kW];
        complex actual_power[kVA];
}
```

Occupants

```
class occupantload {
        int32 number_of_occupants;
        double occupancy_fraction[unit];
        double heatgain_per_person[Btu/h];
}
```

Microwave

```
class microwave {
        // Note: This object is not fully operational.
        double installed_power[kW];
        double standby_power[kW];
        double circuit_split;
        double cycle_length[s];
        enumeration state; //ON, OFF
        double runtime[s];
        double state_time[s];
}
```

Freezer

```
class freezer {
        // Note: This object is not fully operational.
        double size[cf];
        double rated_capacity[Btu/h];
        double temperature[degF];
        double UA[Btu.h/degF];
        double deadband[degF];
        double setpoint[degF];
}
```

Zipload

```
class ZIPload {
   // In addition to residential enduse variables
   double heat_fraction;
   double base_power[kW];
   double power_pf;
   double current_pf;
   double impedance_pf;
   double breaker_val[A];
   double is_240; //true or false
   complex actual_power[kVA];
}
```

- Generic load model that allows adjustment of all power factors and zip fractions.
- Only uses enduse structure (no state model).

Distribution panel

Appliances are automatically assigned to circuits

- `configuration IS220` flags that load is on a 220V circuit
- 110V loads are placed on alternative circuits

Provides for automatic load balancing

Implements general protection from overvoltage

- Each circuit has a breaker with appropriate Amp limit
- Entire house also has an Amp limit
- Breakers have a limit on number of operation

Exercises

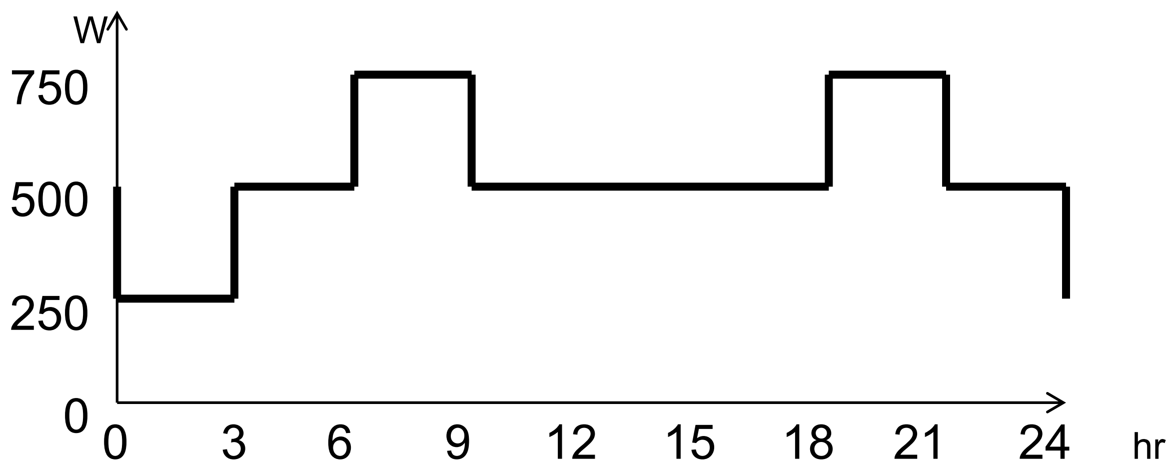
Collect total power consumption for a population of 100 water heaters for one month Hint: use “actual_load”.

Collect the hourly average, stdev, minimum and maximum indoor air temperature a population of 100 default houses for a week Hint: use “air_temperature”.

Collect the minimum and maximum voltages of all nodes in the IEEE 37-bus test model as it converges; Hint: use voltage_A.mag, voltage_B.mag, & voltage_C.mag

Exercises

Define a house with this load shape and record the house plug load.



Define a second house using the same plug load shape but make it pulsed

Exercises

1. Create a climate object that uses Yakima WA tmy2 data. Record the weather data every ten minutes.
2. Repeat Exercise 1, using QUADRATIC and LINEAR interpolations and record at ten minute intervals.
3. Create a climate object that uses the sample csv reader (weather.csv). Record the weather data every hour.

Exercises

4. **Create a house object with all end-uses implemented and plot the power output for a winter week and a summer week.**
5. **Using the same model, plot the indoor temperature for a winter week and a summer week.**
6. **Modify the same house to only contain the HVAC and a zipload object that is driven by a schedule transform (no enduses).**
   a) Record the power output over time.
   b) Modify the ZIP fractions, and observe the difference.
7. **Create a house with two water heaters.**
   a) Make the first waterheater a physical model
   b) Make the second driven by a load_shape.
   c) Compare power output from the water heaters.

   Hint: use a schedule transform for the first and use “loadshape myshape” with “water_demand this.myshape” for the second.