The BLDG Users' Guide

Version 0.1 beta

Kevin J. Kircher and K. Max Zhang

June 30, 2016

Contents

1 Introduction

1.1 What is BLDG?

BLDG is an open-source, control-oriented building simulator implemented in MATLAB. It is intended to eliminate the need to model a building or leave the MATLAB environment before testing an estimation or control algorithm for heating and cooling systems. BLDG was designed to be as simple as possible to use, understand, and extend.

BLDG can simulate a family of simple buildings indexed by a small number of geometric parameters and material properties. Depending on these parameters, the buildings can be thermally heavy or light, can have drafty or tight envelopes, and can be subject to weak or strong solar forcing. The differential equations that govern the buildings' behavior are derived using the same first-principles modeling techniques employed by state-of-the-art building simulators such as EnergyPlus [1] and TRNSYS [2]. In particular, BLDG includes time-varying solar forcing, continuous wall temperature distributions governed by partial differential equations, and nonlinearities arising from thermal radiation and temperature-dependent convection coefficients. BLDG has been empirically validated in hardware; see [3] for details.

At its core, BLDG represents a building mathematically as a nonlinear, time-varying, discrete-time dynamical system. In this representation, the building's behavior is determined by the difference equations

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, 2, \ldots.$$

In the above, k indexes discrete time, x_k is the building state (a list of temperatures), u_k is the control action of the heating or cooling system (either a heat flow, or a supply air temperature and mass flow rate), and w_k is the disturbance acting on the building (the outdoor air temperature, solar irradiance, and heat flows from lights, occupants' bodies, appliances, electronics, *etc.*). The basic purpose of BLDG is to provide the dynamics mappings f_k. This is accomplished by BLDG's central function, `bsim`.

BLDG has a number of other uses. It can instantiate a default building object in one line of code, and allows the user to easily vary the object's geometric parameters and material properties. BLDG can import historical weather data from any of the 1,109 sites in the National Solar Radiation database maintained by the National Renewable Energy Laboratory. [4] BLDG can generate random internal heat source signals that reasonably approximate those of a typical residential or commercial building. It can compute a building's heating and cooling loads in order to help size climate control equipment. Finally, BLDG can plot simulation inputs and outputs.

1.2 What BLDG is *not*

BLDG is not a general-purpose building simulator. It does not compete with, and will never replace, software packages such as EnergyPlus and TRNSYS. These packages are

designed to handle a wide variety of building geometries, construction types, mechanical and electrical systems, and patterns of use. This design philosophy has enabled diverse applications such as comparing the energy-efficiency of architectural designs, predicting the energy savings of building retrofits, sizing heating and cooling systems, analyzing the economics of solar panels, and many more. However, the generality of simulators such as EnergyPlus and TRNSYS necessitates a level of conceptual and computational complexity that renders them opaque to most users.

By contrast, BLDG is designed to capture, as simply as possible, the *essence* of buildings' dynamical behavior. It is designed for researchers, teachers, and students who want to understand complex physical or algorithmic behavior in a transparent, high-fidelity test-bed. Although BLDG can simulate only a small family of simple buildings, in our experience this is often sufficient to enable investigation into open research questions. Several examples can be found in [5].

BLDG is not a *control* toolbox. BLDG cannot learn control-oriented building models from data, estimate the states and parameters of those models online, or implement model-based control schemes. Rather, BLDG is intended to enable the *testing* and *comparison* of system identification, online estimation, and advanced control algorithms. The key value added by BLDG is its capacity to act as a nonlinear, time-varying 'truth-model.' Users are free to test any algorithm they can dream up (and code). That said, we do provide code for a number of candidate algorithms in the `examples` library.

BLDG is not middleware. BLDG does not provide an interface between MATLAB and any other software. BLDG simulates buildings by numerically solving the governing differential equations in MATLAB. Therefore, the only programming knowledge required to use, modify, or extend BLDG is a basic familiarity with MATLAB.

1.3 Other MATLAB building toolboxes

As discussed in §1.1 and §1.2, the scope of BLDG is intentionally narrow. If you are looking for a tool that can simulate large, multizone buildings with complex heating and cooling systems, or a tool that can automatically implement estimation and control algorithms, then BLDG is not for you. Instead, we recommend you look into the Building Resistance-Capacitance Modeling (BRCM) [6] and OpenBuild [7] toolboxes. Both are built on the Building Controls Virtual Test-Bed [8] and MLE+ [9], which together enable communication between MATLAB and EnergyPlus.

BRCM was developed by David Sturzenegger *et al.* in the Automatic Control Laboratory at ETH Zurich. BRCM is intended to enable model predictive control of real, complex buildings. It maps EnergyPlus models into bilinear resistor-capacitor networks, and generates cost and constraint matrices for model predictive control. An empirical demonstration of the use of BRCM for model predictive control can be found in [10].

OpenBuild was developed by Tomasz Gorecki *et al.* in the Automatic Control Laboratory at EPF Lausanne. Like BRCM, OpenBuild maps EnergyPlus models into resistor-capacitor networks and provides tools for model predictive control. Unlike BRCM, however, OpenBuild allows control decisions to be sent from MATLAB to EnergyPlus. This enables researchers to test algorithms on EnergyPlus's nonlinear truth-models. In principle, OpenBuild could be used to study as wide a range of building geometries and HVAC systems as EnergyPlus can model (though some EnergyPlus functionality is not yet supported). OpenBuild has been used to study the demand response potential of commercial buildings in [11].

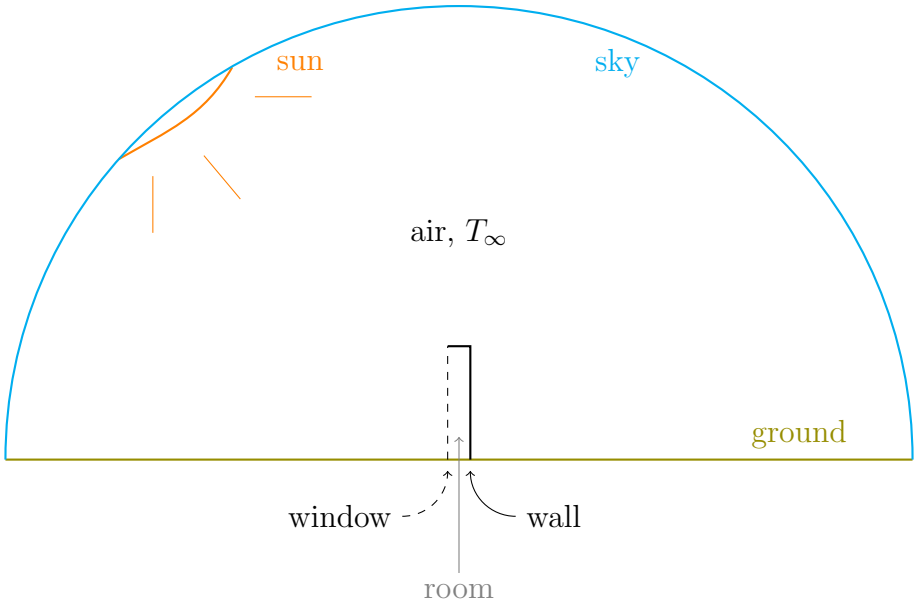

Figure 1: the building and its environment.

2 Simulator overview

As discussed in §1.1, BLDG simulates a family of simple buildings indexed by a small number of parameters. In this section, we briefly describe how these buildings are modeled and simulated in BLDG.

Figure 1 shows the building and its surrounding environment. The sky, ground, and outdoor air are assumed to be isothermal at temperature T_∞. The building is a box-shaped room. Heat transfer through two of the building's vertical sides is assumed to dominate that through the other four sides. One of the dominant sides is a single-pane window. The other is a wall of uniform composition.

Figure 2 shows a zoomed-in schematic of the building. Since the window glass is highly conductive and its thickness l_g is small, it is well-modeled as a single lumped capacitance with temperature T_g (see [12] for details). The wall, on the other hand, may be thick and well-insulated, so its temperature must be modeled as a continuous distribution $\{T(x,t) \mid x \in [0,l],\ t \geq 0\}$, where l is the wall thickness. The indoor air is assumed to be well-mixed with temperature T_a. Heat is transferred through convection, radiation, and the transmission, absorption, and reflection of visible light. The internal source Q_a includes heat flows from people, control systems, lights, and other equipment (appliances, electronics, *etc.*).

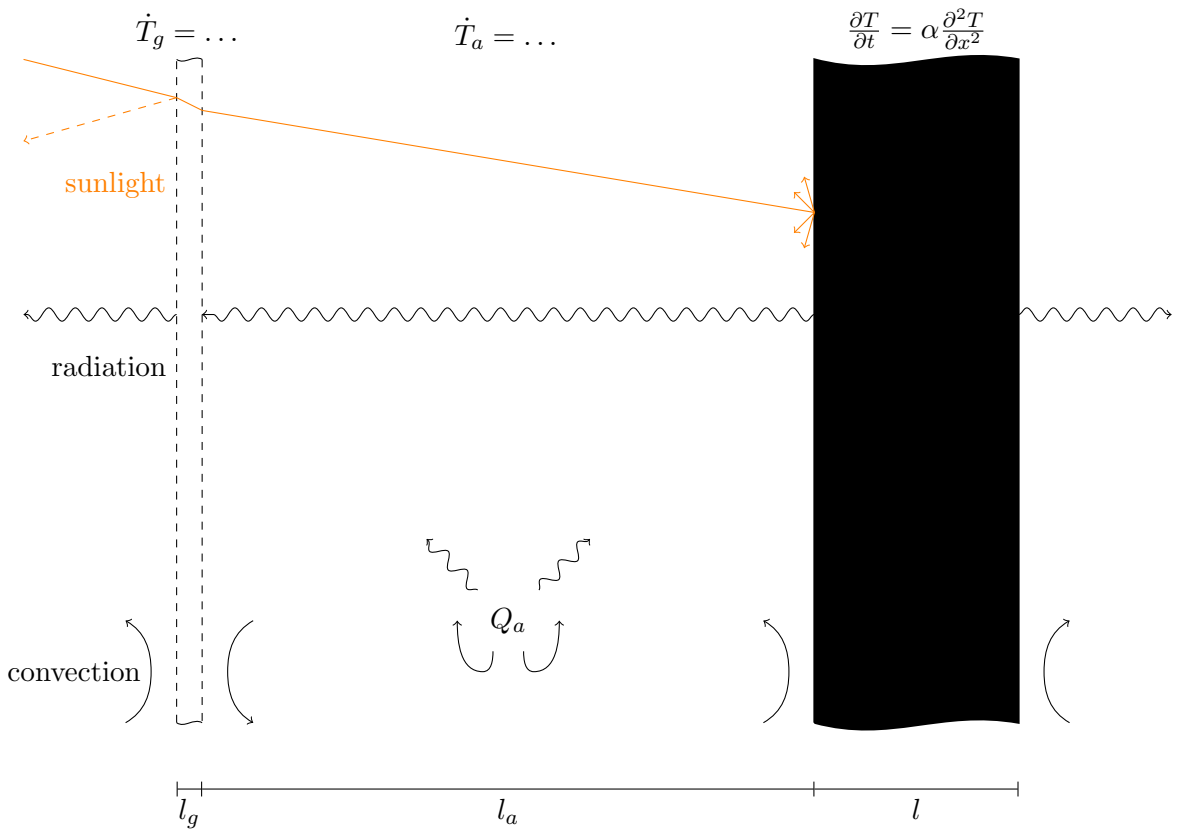

Figure 2: heat transfer mechanisms between the indoor air, window, wall, and environment. Two other effects are included in the model, but not pictured. These are diffuse solar radiation and shortwave radiation from artificial lighting inside the building.

2.1 Governing equations

The wall temperature distribution $\{T(x,t) \mid x \in [0,l],\ t \geq 0\}$ satisfies the heat equation,

$$\begin{aligned} &\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}, \quad T(x,0) = T^0(x) \\ &-k\frac{\partial T}{\partial x}\bigg|_- = q_-, \quad -k\frac{\partial T}{\partial x}\bigg|_+ = q_+. \end{aligned} \tag{1}$$

Here α (m^2/s) and k (W/m·K) are the wall's thermal diffusivity and conductivity, $T^0(\cdot)$ is the initial temperature distribution, and the subscripts $-$ and $+$ indicate the left and right wall surfaces, respectively. The fluxes q_- and q_+ (W/m^2), which are nonlinear and time-varying, include convection with temperature-dependent coefficients, longwave radiation exchange with the window and surrounding environment, and shortwave radiation from the sun and artificial lighting.

The window temperature satisfies

$$C_g \dot{T}_g = A(q_{g-} - q_{g+}), \quad T_g(0) = T_g^0, \tag{2}$$

where C_g (J/K) is the thermal capacitance of the window and A (m^2) is its surface area. As with the wall, the window surface fluxes q_{g-} and q_{g+} include nonlinear convection and radiation. The model also accounts for the reflection, absorption and transmission of visible light by the window.

The indoor air is assumed to be well-mixed, with temperature T_a governed by

$$C_a \dot{T}_a = A(q_{a-} - q_{a+}) + Q_a, \quad T_a(0) = T_a^0, \tag{3}$$

where C_a (J/K) is the thermal capacitance of the air and any material that is isothermal with it (*e.g.*, ducts, dampers, or furniture). The fluxes q_{a-} and q_{a+} couple the air to the window and wall.

By carefully discretizing spatial derivatives, the PDE (1) can be written with $\mathcal{O}((\Delta x)^2)$ truncation error as a system of N coupled ODEs, where N is the number of finite difference nodes in the wall and $\Delta x = l/(N-1)$. These ODEs are coupled to Equations (2) and (3) through q_-, q_{g+}, and q_{a+}, giving a nonlinear, time-varying dynamical system that can be simulated by a MATLAB® stiff ODE solver. The system state is $(T_1, \ldots, T_N, T_g, T_a)$, where T_i is the temperature of the i^{th} wall node.

2.2 Inputs and outputs

In this section, we describe the inputs and outputs of the central BLDG function `bsim`. As depicted in Figure 3, the inputs are an object `b` of class `bldg`, a simulation time span `t`, a matrix `W` of disturbances, an initial state `x0`, and a matrix `U` of controls. The rows of the output matrix `X` contain the state of the building at each time step.

Each row of `W` contains $(T_\infty, I_h, I_\perp^b, Q_p, Q_l, Q_e)$, where T_∞ is the outdoor air temperature, I_h (W/m^2) is the total solar irradiance on a horizontal surface, $I_\perp^b$ (W/m^2) is the beam

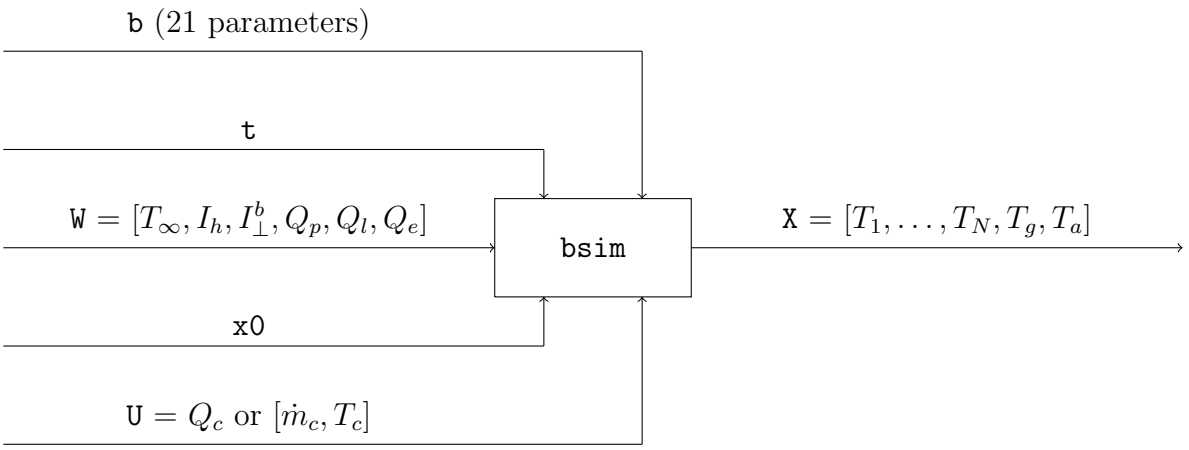

Figure 3: input-output behavior of the main BLDG function `bsim`.

.

irradiance on a surface normal to the sun, and Q_p, Q_l, and Q_e (W) are the internal heat flows from people, lights, and other equipment.

Each row of `U` contains either Q_c (W), the heat supplied by control systems, or $(\dot{m}_c, T_c)$, where $\dot{m}_c$ (kg/s) is the mass flow rate of supply air and T_c is the supply air temperature. In the second case, the heat supplied by control systems is $Q_c = \dot{m}_c c_a (T_c - T_a)$, where c_a (J/kg·K) is the specific heat of air at constant pressure; we discuss this case in §4.3. The function `bsim` can also be called with the syntax

```
[X,Q] = bsim(b,t,W,x0);
```

In this case, the returned vector `Q` contains the heat flows required to perfectly regulate the air temperature at its initial value. We discuss this case in §4.4.

2.3 Further reading

If you are interested in further details about the model, we suggest reading the journal article [3]. This article includes

- a list of all modeling assumptions,
- a description of the numerical methods employed by BLDG,
- validation of the model and numerical methods through a series of experiments performed in a laboratory test-bed, and
- an analysis of the sensitivity of the model outputs to the dominant parameters.

3 Installation

BLDG has been tested on Mac OS X 10.9 and Windows 7, and is known to be compatible with MATLAB R2014a or later. We suspect that BLDG is also compatible with most UNIX variants and MATLAB versions R2009a or later. If you encounter compatibility issues, please report them by email.

Mac and Windows installation guide:

1. Start MATLAB.
2. If you have an older version of BLDG installed on your machine, remove the `bldg` folder *and all subfolders* from the MATLAB path. Once this is done, we recommend renaming the `bldg` folder to something like `bldg_old`, but not deleting it until the new version has been successfully installed.
3. Download the latest BLDG distribution as a .zip file from the GitHub page.
4. Unpack the .zip file into the desired location. This generates a folder called `bldg`.
5. In MATLAB, navigate to the `bldg` folder and run `bldgSetup` from the command window. This will add the required files to the MATLAB path, then test BLDG.

On UNIX machines, `bldgSetup` may fail to update the MATLAB path due to a lack of permissions. In this case, step 5 will need to be replaced by manually adding the subfolder `functions` of the main folder `bldg` to the MATLAB path, then saving the path.

4 Quick start

Once BLDG has been successfully installed, you can use it within any MATLAB script or function or from the command line. In this section, we demonstrate BLDG's syntax and capabilities through a few simple examples. We encourage the reader to follow along in MATLAB by running the script `quickStart` within the `examples` directory.

4.1 Defining a building

A default building object can be defined by typing

```
b = bldg;
```

Displaying this object by entering `disp(b)` (or just `b`, with no semicolon) produces the following output.

```
Object of class BLDG.

Geometric parameters:
  Wall thickness: l = 0.2 m.
  Glass thickness: lg = 0.0032 m.
  Room length: la = 7 m.
  Wall/glass surface area: A = 21 m^2.
  Wall azimuth (0 = north): gam = 3.14 rad.
  Building latitude (pi/2 = north pole): phi = 0.712 rad.

Material properties:
  Wall diffusivity: a = 7.4e-08 m^2/s.
  Wall conductivity: k = 0.12 W/(m*K).
  Wall longwave emissivity: eps = 0.9.
  Wall shortwave absorptivity: as = 0.5.
  Glass capacitance: Cg = 1.5e+05 J/K.
  Glass longwave emissivity: epsg = 0.2.
  Glass index of refraction: ng = 1.53.
  Glass attenuation coefficient: mug = 19.6 m^(-1).
  Room capacitance: Ca = 3.62e+06 J/K.

Internal heat source properties:
  Outdoor air infiltration mass flow rate: mdot = 0.05 kg/s.
  Convective fraction of heat from control systems: zc = 1.
  Convective fraction of occupant body heat: zp = 0.4.
  Convective fraction of heat from lighting: zl = 0.75.
  Convective fraction of heat from miscellaneous equipment: ze = 0.2.
  Lighting efficiency: eta = 0.2.
```

The properties in `b`, such as the wall conductivity `b.k`, can easily be redefined:

```
b.k = 0.1;
```

If a physically meaningless value is specified, however, BLDG will throw an error. For example, attempting to specify the wall thickness `b.l = -1` results in the following error:

```
Error using bldg/set.l (line 71)
Wall thickness must be positive.
```

4.2 Simulating step responses

Now that we've defined a building, let's simulate its response to a sudden drop in the outdoor air temperature from 20 to 0 degrees Celsius. This can be accomplished with the `bsim` function, whose input-output behavior is described in §2.2.

The first step is to define a simulation time span. The following code creates a five-day time span with 15-minute time steps.

```
dt = 15*60;                                    % time step in seconds
t = 0:dt:5*24*3600;
```

Note that all times in BLDG must be specified in units of seconds.

Next, we define the disturbance and control signals. For simplicity, we set all signals to be zero except the outdoor air temperature T_∞, which forms the first column of the disturbance matrix `W`.

```
nt = length(t);
nw = 6;                                        % disturbance dimension
nu = 1;                                        % control dimension
W = zeros(nt,nw);
W(:,1) = 273;                                  % outdoor air temperature
W(1:round(nt/10),1) = 293;
U = zeros(nt,nu);
```

In BLDG, all temperatures are in units of Kelvin, so the line `W(:,1) = 273;` sets all the outdoor air temperatures to 273 Kelvin (0 Celsius). The subsequent line changes the first 10% of the outdoor air temperatures to 293 Kelvin (20 Celsius).

Our last steps are to define the number `N` of finite difference nodes in the wall, and to set the initial temperatures of the glass, indoor air, and the `N` wall nodes to 293 Kelvin (20 Celsius).

```
N = 50;
x0 = 293*ones(N+2,1);
```

We can now simulate the building's response to a step change in the outdoor air temperature from 20 Celsius to 0 Celsius.

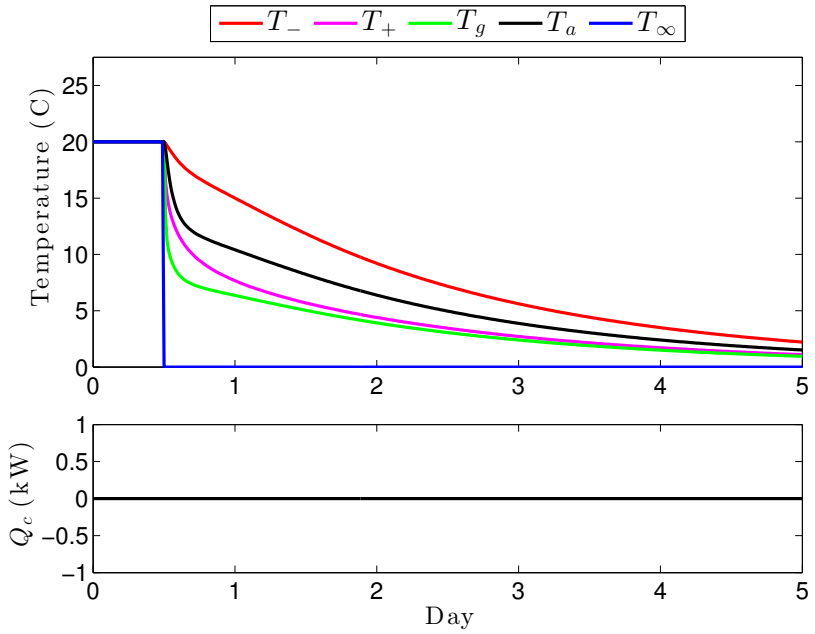

Figure 4: response of the wall surface temperatures T_- and T_+, the glass temperature T_g, and the indoor air temperature T_a to the step change in the outdoor air temperature T_∞ in §4.2.

```
X = bsim(b,t,W,x0,U);
plotResults(t,X,W,U)
```

Figure 4 shows the simulation outputs.

It is also easy to simulate the building's response to a step change in the control heat flow. To do this, we simply redefine the disturbance and control signals and simulate again.

```
W(:,1) = 293;
U(round(nt/10):end) = 5e3;
X = bsim(b,t,W,x0,U);
plotResults(t,X,W,U)
```

Figure 5 shows the results.

4.3 Simulating a forced-air heating system

In the example in §4.2, we controlled the heat flow Q_c directly. As discussed in §2.2, however, `bsim` can also simulate forced-air heating and cooling systems, where the supply air mass flow rate $\dot{m}_c$ and temperature T_c are controlled. The heat flow is then $Q_c = \dot{m}_c c_a (T_c - T_a)$, where c_a (J/kg·K) is the specific heat of air at constant pressure.

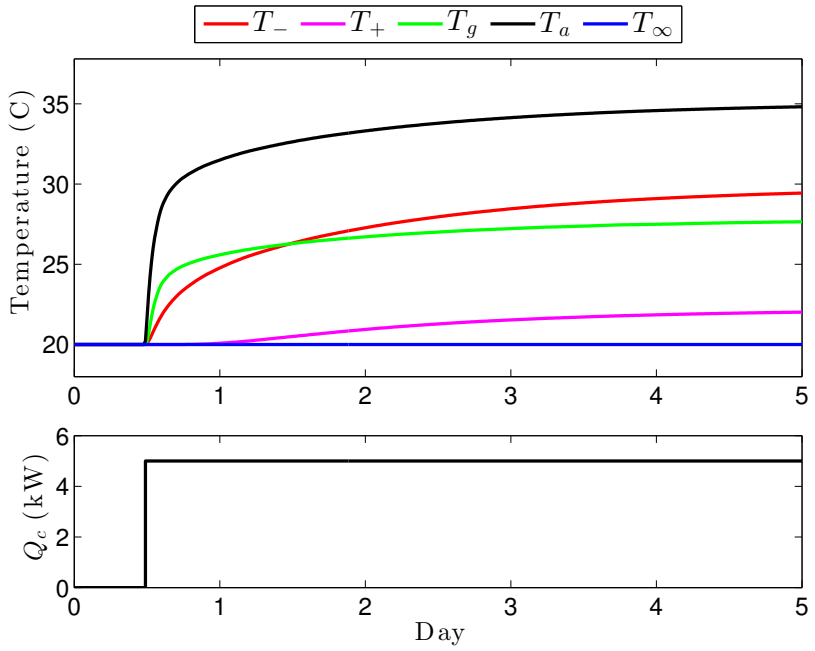

Figure 5: response of the wall surface temperatures T_- and T_+, the glass temperature T_g, and the indoor air temperature T_a to the step change in the control heat flow Q_c in §4.2.

In this example, we simulate the building's response to a sinusoidally-varying supply air mass flow rate and constant supply air temperature. The first step is to redefine the control dimension and set the supply air temperature to a constant, in this case 325 Kelvin (52 Celsius).

```
nu = 2;
U = zeros(nt,nu);
U(:,2) = 325;
```

Next, we define the sinusoidal mass flow rate signal, which varies between 0 and 0.04 kg/s with a period of four hours.

```
U_mean = 0.02; U_amp = U_mean; U_period = 4*3600;
U(:,1) = U_mean + U_amp*sin((2*pi/U_period)*t)';
```

We can now simulate the building's response:

```
X = bsim(b,t,W,x0,U);
plotResults(t,X,W,U)
```

Figure 6 shows the simulation outputs.

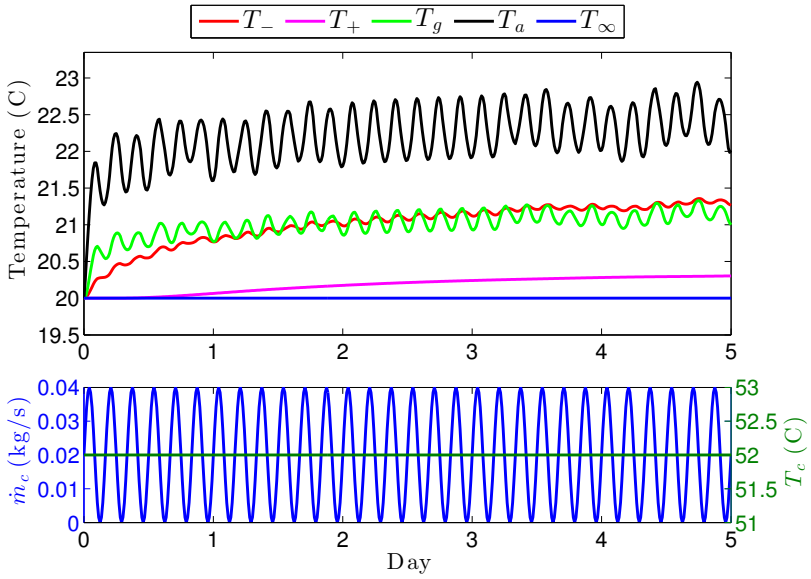

Figure 6: response of the wall surface temperatures T_- and T_+, the glass temperature T_g, and the indoor air temperature T_a to the sinusoidally-varying supply air mass flow rate $\dot{m}_c$ in §4.3.

4.4 Calculating peak heating and cooling loads

In the examples in §4.2 and §4.3, we defined the disturbance signals by hand. However, BLDG also provides tools to import historical weather data and generate random internal heat flows that approximate those of a typical residential or commercial building. In this example, we use those tools to calculate the building's peak heating and cooling loads over a six-month time span. These calculations also use `bsim`'s ability to simulate 'perfect control', where the indoor air temperature is regulated exactly at a specified value.

We begin by importing a year of historical weather data:

```
dt = 15*60;                                    % time step in seconds
[b,weather] = importWeather(b,'NYC_TMY3.csv',dt);
```

The Typical Meteorological Year 3 file `NYC_TMY3.csv` contains one year of historical New York City weather data at hourly resolution. The function `importWeather` interpolates the hourly data to the desired 15-minute resolution. Weather files for about one thousand other locations can be freely obtained from the National Renewable Energy Laboratory database.

Next, we generate the heat flows from internal sources (other than heating and cooling systems) to approximate those in a residential building:

```
gains = generateGains(b,weather.tw,'residential');
```

The function `generateGains` generates one year of random internal gains. Its temporal resolution matches that the weather data time span `weather.tw`. Plausible internal gains for

a commercial building can also be generated by calling `generateGains` with `'commercial'` as the third argument.

We now define the simulation time span. This is done by specifying the initial day number `nd0` (an integer between 1 and 365, with 1 representing January 1), and the initial and final times `t0` and `tf` (both measured in seconds since midnight on day `nd0`). The following code defines a six-month simulation time span starting at midnight on January 18.

```
nd0 = 18;
t0 = 0; tf = 6*30*24*3600;
t = getTiming(weather.tw,nd0,t0,tf);
```

Next, we find an initial state `x0` from which to start the building:

```
Ts = 293;                                          % temperature setpoint
x0 = precondition(b,t,weather,gains,N,Ts);
```

The `precondition` function simulates the building for the two weeks leading up to time `t(1)`, under the appropriate disturbances, with the indoor air temperature regulated exactly at the setpoint `Ts`.

Finally, we pack the weather and internal gains into a disturbance matrix `W`, simulate the building, and compute the peak loads:

```
W = getDisturbances(weather,gains,t);
[X,U] = bsim(b,t,W,x0);
U_max = max(U);
U_min = min(U);
```

When called with the syntax `[X,U] = bsim(b,t,W,x0)`, `bsim` returns a vector `U` of control heat flows required to perfectly regulate the indoor air temperature at its initial value `x0(end)`. In other words, `U` contains the building's instantaneous heating load (or if negative, cooling load) at every time step. In this example, `U_max` = 11 kW and `U_min` = −9.7 kW. Therefore, the building's heating system should be sized to provide at least 11 kW of heat to the building, and its cooling system should be sized to remove at least 9.7 kW of heat from the building.

4.5 Simulating closed-loop operation

The examples in §4.2 and §4.3 simulated the building's *open-loop* behavior: The control signals were decided without taking into account any information about the building state. BLDG can also be used to simulate the buildings' *closed-loop* behavior, with control decisions based on feedback about the building state. In this example, we simulate perhaps the simplest feedback controller, a thermostat. The simulated thermostat maintains the indoor air temperature near the stepoint `Ts` = 293 Kelvin by turning the heat on at full capacity when the indoor air temperature drops below `T_low`= 291 Kelvin, and off when it exceeds `T_high`= 295 Kelvin.

We begin by defining the heater capacity using the peak heating load `U_max` from §4.4.

```
Qc0 = 1.5*U_max
```

Here we have oversized the heating system by 50%. This amount of oversizing is on the low end for both residential and commercial buildings.

Next, we define a one-day simulation time span starting at midnight on January 18, and extract the appropriate disturbances from the `weather` and `gains` objects.

```
nd0 = 18;
t0 = 0; tf = 24*3600;
t = getTiming(weather.tw,nd0,t0,tf);
W = getDisturbances(weather,gains,t);
```

We then initialize the state matrix `X` and control vector `U`.

```
nt = length(t);
X = zeros(nt,N+2);
X(1,:) = x0';
nu = 1;
U = zeros(nt,nu);
```

Next, we loop through the simulation time span, with `bsim` playing the role of the nonlinear, time-varying dynamics functions f_k in the difference equations $x_{k+1} = f_k(x_k, u_k, w_k)$.

```
for k = 1 : nt-1
  % Let the system evolve.
  x_new = bsim(b,t(k:k+1),W(k,:),X(k,:),U(k));

  % Decide the thermostat's control action.
  Ta = x_new(end);
  if Ta < T_low
    U(k+1) = Qc0;
  elseif Ta > T_high
    U(k+1) = 0;
  else
    U(k+1) = U(k);
  end

  % Store the data from this iteration.
  X(k+1,:) = x_new';
end
```

As with most MATLAB operations involving `for` loops, this simulation is slow. In fact, it is about twenty times slower than an open-loop simulation over the same time span. However, this simulation method allows any feedback control scheme to be simulated – even a complex, optimization-based method such as model predictive control. In our experience with model

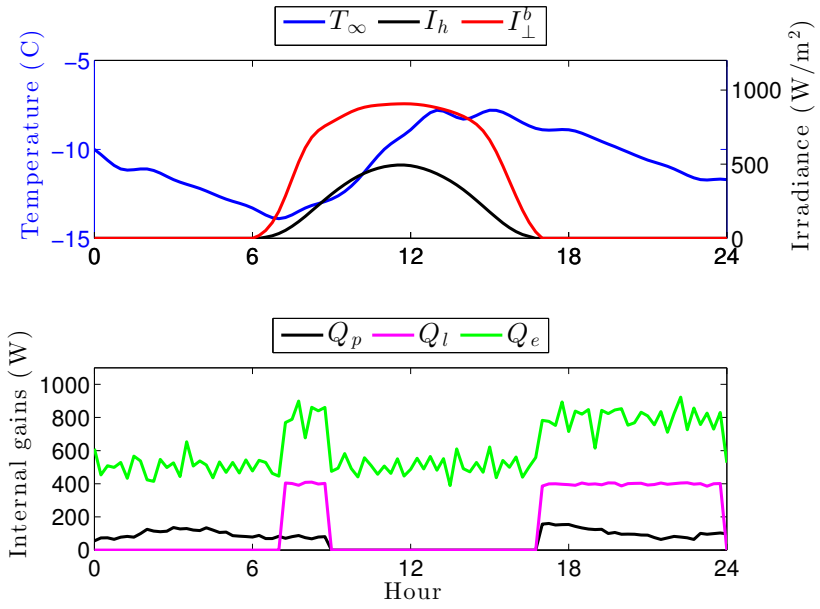

Figure 7: disturbances on the closed-loop simulation day from §4.5. The top plot shows the outdoor air temperature T_∞, the total solar irradiance I_h on a horizontal surface, and the beam solar irradiance $I^b_\perp$ on a surface normal to the sun. The bottom plot shows the heat flows Q_p, Q_l, and Q_e from people, lights, and miscellaneous equipment, which model a single-occupant home on a weekday.

predictive control, the computational overhead added by `bsim` is small compared to the time spent in optimization (see §IV-C of [5] for details).

Finally, we plot the simulation inputs and outputs.

```
plotInputs(t,W)
plotResults(t,X,W,U,T_low,T_high)
```

Figures 7 and 8 show the plots.

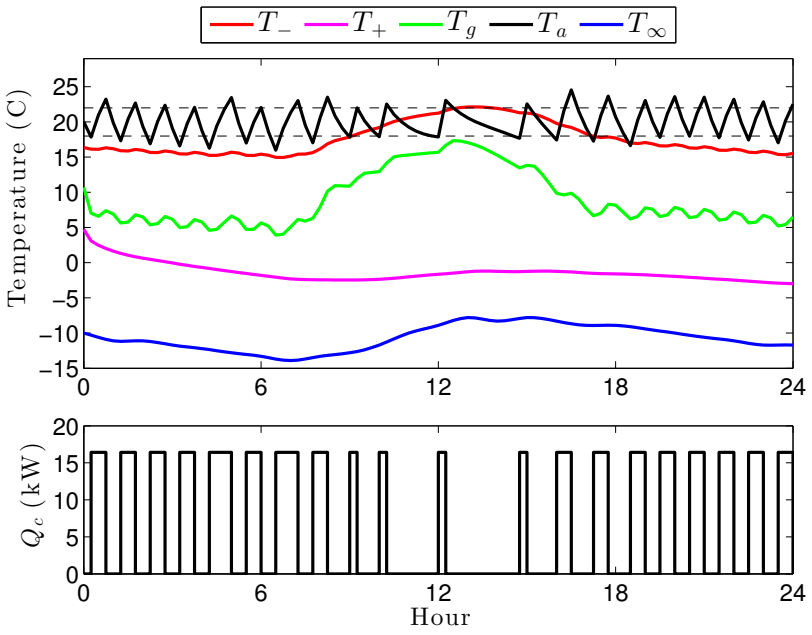

Figure 8: the wall surface temperatures T_- and T_+, the glass temperature T_g, and the indoor air temperature T_a, in the closed-loop simulation from §4.5. The heat flow Q_c is controlled by a thermostat, with deadband shown in dashed lines in the top plot.

5 Support

This document is intended as the first line of support for users. If it doesn't address your question, then the references [3] and [5] may provide some illumination. We are also happy to provide email support. We are particularly interested in finding and fixing bugs, but we'd also love to hear about functionality that you'd like to see added, or concepts or tools that have not been clearly explained.

6 License

BLDG is free and open source. It is provided under the highly permissive MIT license.

7 Citing BLDG

BLDG was designed to aid research, teaching, and learning about energy efficiency in buildings in general, and building climate control algorithms in particular. If you are using BLDG for these (or other) purposes, please let us know by email! We are genuinely curious about how BLDG is used now, and how it might be used in the future if new functionality were added.

If you are using BLDG to facilitate published research, we would appreciate a formal mention of our work. For example, a citation might look like this:

> Algorithm (3) was tested in the MATLAB BLDG toolbox, which acts as a non-linear, time-varying truth-model. [3, 5]

In the above, references [3, 5] are the following:

3. Kircher, K.J. & Zhang, K.M. *Testing building controls with the BLDG toolbox* in *American Control Conference (ACC)* (2016).

5. Kircher, K.J., Schaefer, W., & Zhang, K.M. The simplest building. *Energy and Buildings* **TBD** (2016).

Here are the corresponding BiBTeX entries:

```
@conference{bldg2016,
  Author = {Kircher, K. J. and Zhang, K. Max},
  Booktitle = {American Control Conference (ACC)},
  Title = {Testing building controls with the {BLDG} toolbox},
  Year = 2016
}

@article{sb2016,
  Author = {Kircher, K. J. and Schaefer, W. and Zhang, K. Max},
  Journal = {Energy and Buildings},
  Title = {The simplest building},
  Volume = {TBD},
  Year = 2016
}
```

8 Acknowledgments

BLDG was designed and implemented by Kevin J. Kircher while a PhD student in the Sibley School of Mechanical and Aerospace Engineering at Cornell University, under the advisement of Professor K. Max Zhang. Kevin is grateful to the Hydro Research Foundation and the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy for financial support.

BLDG was empirically validated in hardware designed and built by Walter Schaefer and Dr. Justin Dobbs, with assistance from Sean Hidaka. We thank them kindly for their help.

We are also grateful to Dr. Michael Grant of CVX Research and Professor Stephen Boyd of Stanford University. They created the CVX toolbox, which models and solves convex optimization problems in MATLAB. Much of the BLDG toolbox organization, including the format of this document, was based heavily on CVX.

9 References

1. Crawley, D. *et al.* EnergyPlus: creating a new-generation building energy simulation program. *Energy and Buildings* **33,** 319–331 (2001).
2. Klein, S. *et al. TRNSYS 16 - A TRaNsient system simulation program, user manual* Solar Energy Laboratory, University of Wisconsin-Madison (Madison, Wisconsin, 2004).
3. Kircher, K. J., Schaefer, W. & Zhang, K. M. The simplest building. *Energy and Buildings* **TBD** (2016).
4. Wilcox, S. & Marion., W. *User's Manual for TMY3 Data Sets* tech. rep. NREL/TP-581-43156 (National Renewable Energy Laboratory, Apr. 2008).
5. Kircher, K. J. & Zhang, K. M. *Testing building controls with the BLDG toolbox* in *American Control Conference (ACC)* (2016).
6. Sturzenegger, D., Gyalistras, D., Semeraro, V., Morari, M. & Smith, R. *BRCM Matlab Toolbox: Model Generation for Model Predictive Building Control* in *American Control Conference* (2014), 1063–1069.
7. Gorecki, T., Qureshi, F. & Jones, C. *OpenBuild: An integrated simulation environment for building control* in *IEEE Conference on Control Applications (CCA)* (2015), 1522–1527.
8. Wetter, M. Co-Simulation of Building Energy and Control Systems with the Building Controls Virtual Test Bed. *Journal of Building Performance Simulation* **4,** 185–203 (2011).
9. Bernal, W., Behl, M., Nghiem, T. & Mangharam, R. *MLE+: A Tool for Integrated Design and Deployment of Energy Efficient Building Controls* in *Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings* (2012), 123–130.
10. Sturzenegger, D. *et al. Model Predictive Control of a Swiss Office Building* in *11th RHEVA World Congress Clima* (2013).
11. Qureshi, F., Gorecki, T. & Jones, C. *Model predictive control for market-based demand response participation* in *World Congress of the International Federation of Automatic Control* (2014).
12. Kircher, K. J. & Zhang, K. M. On the lumped capacitance approximation accuracy in RC network building models. *Energy and Buildings* **104,** 454–462 (2015).