```
function [T, X, Y, U, ctrlState] = closedLoopSimulationComplete(x0, tspan, D, p, ...
    simModel, observationModel, ctrlAlgorithm, ...
    ctrlPar, ctrlState0, simMethod, tzero, haltingiter, idxbo, rampingfunction, ...
    dg, dt, gridTime, mealTime, opts)
% CLOSEDLOOPSIMULATION Simulate a closed-loop control algorithm.
%
% DESCRIPTION:
% Perform a closed-loop simulation of a model-based control algorithm for
% given initial condition, control intervals, disturbance variables,
% parameters, simulation model, observation model, and control algorithm
% (including hyperparamters, control model, control model paramters, and
% initial controller state).
%
% REQUIRED PARAMETERS:
%   x0                  - initial state↙
(dimension: nx    )
%   tspan               - boundaries of the control intervals↙
(dimension: N+1   )
%   D                   - disturbance variables for each control interval↙
(dimension: nd x N)
%   p                   - parameters↙
(dimension: np    )
%   simModel            - simulation model          (function handle)
%   observationModel    - observation model         (function handle)
%   ctrlAlgorithm       - control algorithm         (function handle)
%   ctrlPar             - controller parameters
%   ctrlState0          - initial controller state↙
(dimension: nc)
%   simMethod           - simulation method         (function handle)
%   opts                - an options structure (must contain the field Nk)
%   tzero               - time where ramping function is 0
%   haltingiter         - time where integration is halted has basal is ramped
%   dg, dt              - parameters used in GRID
%   gridTime            - time period parsed to GRID at every time step
%   mealTime            - time period in which only 1 meal is allowed
%
% RETURNS:
%   T - boundaries of control intervals (=tspan)    (dimension:      N+1)
%   X - the states in the simulation model          (dimension: nx x N+1)
%   X - the observed variables                      (dimension: ny x N+1)
%   U - the computed manipulated inputs             (dimension: nu x N  )
%   ctrlState - matrix of controller states         (dimension: nc x N+1)
%
% AUTHORS
% Tobias K. S. Ritschel
% Asbjørn Thode Reenberg
% John Bagterp Jørgensen
%
% CHANGED BY
%
% Frederik Nagel
% Laurits Fog Balstrup
% Morten M. Christensen

% Initial time
t0 = tspan(1);

% Observed variables
y0 = observationModel(t0, x0, p);

% Integration halt time during bolus titration
tpause = 0;
```

```
% Determine the number of manipulated inputs
uDummy = ctrlAlgorithm(t0, NaN, NaN, ctrlPar, ctrlState0, tzero, tpause, haltingiter,↙
rampingfunction);

% Number of states and manipulated inputs
nx = numel(x0);
ny = numel(y0);
nu = numel(uDummy);
nc = numel(ctrlState0);

% Number of control intervals
N = numel(tspan)-1;

% Number of time steps in each control interval
Nk = opts.Nk;

% Allocate memory
T = zeros( 1, N+1);
X = zeros(nx, N+1);
Y = zeros(ny, N+1);
U = zeros(nu, N  );
ctrlState = zeros(nc, N+1);

% Store initial condition
T(   1) = t0;
X(:, 1) = x0;
Y(:, 1) = y0;
ctrlState(:, 1) = ctrlState0;

% Copy initial condition
tk = t0;
xk = x0;
yk = y0;

% Initialize GRID
GRID = zeros(1,gridTime);

Dest = zeros(1,length(tspan));

% Maximum meal sizes allowed
maxMeal = 2000;

% bolus scalar
alpha = 0.25;
beta = 0.5;
gamma = 0.1;

% time period used for computing Gsc derivative
TdG = 6;

for k = 1:N
    % Times
    tkp1 = tspan(k+1);

    % Controller state
    ctrlStatek = ctrlState(:, k);

    % Disturbance variables
    dk = D(:, k);

    if k > gridTime
```

```
        [~,~,GRID]=GridAlgo(Y(k-gridTime:k),dg,dt,[],tspan(k-gridTime:k));
    end

    % If meal is detected and no meal has been detected in past
    % gridTime
    if k > mealTime && nnz(GRID) > 0 && nnz(Dest(k-mealTime:k)) == 0
        dkest = 1;
    else
        dkest = 0;
    end

    Dest(k) = dkest;

    % If meal is detected, halt integration for some iterations and
    % estimate bolus
    if dkest ~= 0
        dYmean = mean(Y(k+1-TdG:k)-Y(k-TdG:k-1));

        tpause = haltingiter;

        ubok = min(maxMeal,alpha*Y(k)*haltingiter*max([0,beta+gamma*dYmean]));
    else
        ubok = 0;
    end

    if yk < ctrlPar(5)
        ubok = 0;
    end

    % Decrement tpause until 0
    if tpause ~= 0
        tpause = tpause - 1;
    end

    % Compute manipulated inputs
    [uk, ctrlStatekp1] = ctrlAlgorithm(tk, yk, dk, ctrlPar, ctrlStatek, tzero,↙
tpause, haltingiter, rampingfunction);

    % Set optimal bolus
    uk(idxbo) = ubok;

    % Time interval
    tspank = linspace(tk, tkp1, Nk+1);

    % Solve initial value problem

    [Tk, Xk] = simMethod(simModel, tspank, xk, [], uk, dk, p);

    % States at the next time step
    tkp1 = Tk(end   );
    xkp1 = Xk(end, :)';

    % Observed variables at the next time step
    ykp1 = observationModel(tkp1, xkp1, p);

    % Store solution
    T(   k+1) = tkp1;
    X(:, k+1) = xkp1;
    Y(:, k+1) = ykp1;
    U(:, k  ) = uk;
    ctrlState(:, k+1) = ctrlStatekp1;
```

```
    % Update initial condition
    tk = tkp1;
    xk = xkp1;
    yk = ykp1;
end
```