Exercise 6 - Models of Population Dynamics and Model Design using Tank Systems

Models from the biology and population dynamics lecture, as well as tank models for Model design lecture.

1 Population Growth Model

* P - population size = number of individuals in a population
* der(P) - population change rate, change per time unit
* g - growth factor of population (e.g. % births per year)
* d - death factor of population (e.g. % deaths per year)

```
class PopulationGrowth
  parameter Real g = 0.04    "Growth factor of population";
  parameter Real d = 0.0005  "Death factor of population";
  Real           P(start=10) "Population size, initially 10";
equation
  der(P) = (g-d)*P;
end PopulationGrowth;
```

Change the initial population size and growth and death factors to get an exponentially decreasing population

```
simulate(PopulationGrowth, stopTime=100)
```

```
plot(P)
```

2 Lotka-Volterra Fox and Rabbit Model

* R = rabbits = size of rabbit population
* F = foxes = size of fox population
* der(R) = der(rabbits) = change rate of rabbit population
* der(F) = der(foxes) = change rate of fox population
* gr = g_r = growth factor of rabbits
* df = d_f = death factor of foxes
* drf = d_rf = death factor of rabbits due to foxes
* gfr = g_rf = growth factor of foxes due to rabbits and foxes

```
class LotkaVolterra
  parameter Real g_r =0.04    "Natural growth rate for rabbits";
  parameter Real d_rf=0.0005  "Death rate of rabbits due to foxes";
  parameter Real d_f =0.09    "Natural deathrate for foxes";
  parameter Real g_fr=0.1     "Efficency in growing foxes from
rabbits";
  Real     rabbits(start=700) "Rabbits,(R) with start population 700";
  Real     foxes(start=10)    "Foxes,(F) with start population 10";
equation
  der(rabbits) = g_r*rabbits - d_rf*rabbits*foxes;
  der(foxes)   = g_fr*d_rf*rabbits*foxes - d_f*foxes;
end LotkaVolterra;
```

Change the death and growth rates for foxes and rabbits, simulate, and observe the effects:

```
simulate(LotkaVolterra, stopTime=3000)
```

```
plot({rabbits, foxes}, xrange={0,1000})
```

3Tank Systems for Object-Oriented Modeling

Tank system including source and PI controller:

Exercises:
* Replace the PIcontinuous controller by the PIdiscrete controller and simulate. (see also the book, page 461)
* Create a tank system of 3 connected tanks and simulate.

```
model TankPI
  LiquidSource            source(flowLevel=0.02);
  PIcontinuousController piContinuous(ref=0.25);
  Tank                    tank(area=1);
equation
  connect(source.qOut, tank.qIn);
  connect(tank.tActuator, piContinuous.cOut);
  connect(tank.tSensor, piContinuous.cIn);
end TankPI;
```

The basic tank:

```
model Tank
  ReadSignal  tSensor   "Connector, sensor reading tank level (m)";
  ActSignal   tActuator "Connector, actuator controlling input flow";
  LiquidFlow  qIn       "Connector, flow (m3/s) through input valve";
  LiquidFlow  qOut      "Connector, flow (m3/s) through output valve";
  parameter Real area(unit="m2")       = 0.5;
  parameter Real flowGain(unit="m2/s") = 0.05;
  parameter Real minV=0, maxV=10; // Limits for output valve flow
  Real h(start=0.0, unit="m") "Tank level";
equation
  assert(minV>=0,"minV - minimum Valve level must be >= 0 ");//
  der(h)      = (qIn.lflow-qOut.lflow)/area;   // Mass balance
equation
  qOut.lflow  = LimitValue(minV,maxV,-flowGain*tActuator.act);
  tSensor.val = h;
end Tank;
```

Connectors and Sources:

```
connector ReadSignal "Reading fluid level"
  Real val(unit="m");
end ReadSignal;

connector ActSignal  "Signal to actuator
 for setting valve position"
  Real act;
end ActSignal;

connector LiquidFlow  "Liquid flow at inlets or outlets"
  Real lflow(unit="m3/s");
end LiquidFlow;

model LiquidSource
  LiquidFlow qOut;
  parameter  flowLevel = 0.02;
equation
  qOut.lflow = if time>150 then 3*flowLevel else flowLevel;
end LiquidSource;
```

Controllers:

```
partial model BaseController
  parameter Real Ts(unit = "s") = 0.1"Time period between discrete
samples";
  parameter Real K = 2"Gain";
  parameter Real T(unit = "s") = 10"Time constant";
  ReadSignal cIn"Input sensor level, connector";
  ActSignal  cOut"Control to actuator, connector";
  parameter Real ref"Reference level";
  Real error"Deviation from reference level";
  Real outCtr"Output control signal";
equation
  error = ref - cIn.val;
  cOut.act = outCtr;
end BaseController;
```

PIcontinuous

```
model PIcontinuousController
  extends BaseController(K=2,T=10);
  Real  x  "State variable of continuous PI controller";
equation
  der(x) = error/T;
  outCtr = K*(error+x);
end PIcontinuousController;
```

PIDcontinuous

```
model PIDcontinuousController
  extends BaseController(K = 2, T = 10);
  Real  x;
  Real  y;
equation
  der(x) = error/T;
  y      = T*der(error);
  outCtr = K*(error + x + y);
end PIDcontinuousController;
```

PIdiscrete

```
model PIdiscreteController
  extends BaseController(K = 2, T = 10);
  discrete Real x;
equation
  when sample(0, Ts) then
    x = pre(x) + error * Ts / T;
    outCtr = K * (x+error);
  end when;
end PIdiscreteController;
```

Tank System with continuous PID Controller

```
model TankPID
  LiquidSource            source(flowLevel=0.02);
  PIDcontinuousController pidContinuous(ref=0.25);
  Tank                    tank(area=1);
equation
  connect(source.qOut, tank.qIn);
  connect(tank.tActuator, pidContinuous.cOut);
  connect(tank.tSensor, pidContinuous.cIn);
end TankPID;
```

```
simulate(TankPID, startTime=0, stopTime=1)
```

Two connected tanks:

```
model TanksConnectedPI
  LiquidSource  source(flowLevel=0.02);
  Tank          tank1(area=1), tank2(area=1.3);;
  PIcontinuousController piContinuous1(ref=0.25), piContinuous2
(ref=0.4);
equation
  connect(source.qOut,tank1.qIn);
  connect(tank1.tActuator,piContinuous1.cOut);
  connect(tank1.tSensor,piContinuous1.cIn);
  connect(tank1.qOut,tank2.qIn);
  connect(tank2.tActuator,piContinuous2.cOut);
  connect(tank2.tSensor,piContinuous2.cIn);
end TanksConnectedPI;
```

```
simulate(TankPID, startTime=0, stopTime=1)
```