```
unit tk_td;

interface
uses math,probabilities,AbstractMtxVec,statrandom,mtxexpr, Math387, RndGenerators;
type
   tdose_metric=(Ext_Conc,Int_Conc,Scal_Int_Conc,Scal_Dam,NN);
   tassumption=(Without,IT,SD);
   tdist=(exponential,loglogistic,lognormal,weibull);
   Tdoublearray = array of double;
   tfok=record
    kin, kout:double;
    d_cint,cint:double;
    end;
   tsok=record
     k11,k12,k21,k22:double;
     cint1,cint2,d_cint1,d_cint2:double;
    end;
   tTox_dataset=record
     timepoint:integer;
     concentration:double;
     cint:double;
     damage:double;
     M:double;
     hazard:double;
     survival:double;
    end;
    Ttime_series=array of ttox_dataset;
   tGUTS = record
     //Statevariables
     CW                    :double;
     Cint,Cbiov,CintSc     :double;
     Damage,DamageScaled   :double;
     M, M_old              :double; //Dose metric
     Cum_Hazard            :double; //Cumulative hazard for individual with thres z
     Survival              :double;
     //Rates
     hb,h                 :double; //Background hazard rate, Hazard rate for individual with
     thres z
     //Parameters
     firstorderkinetic:boolean;
     fok:tfok;
     secondorderkinetic:boolean;
     sok:tsok;
     metabolite:boolean;
     scaled_uptake         :boolean; //Uptake of the scaled concentration is known
     kin_scaled            :double; //Uptake rate of teh scaled internal concentration
     ke,kk,kr,z,pz         :double; //Elinmination rate, Killing rate, Recovery rate,
     Threshold for effects
     noi                   :integer; //Number of individuals for the distribution
     calibration           :boolean;
     alpha,beta            :double;
     dose_metric           :tdose_metric;
     assumption            :Tassumption;
     distribution          :tdist;
     sizedependtk          :boolean;
     length,weight         :double;
     //Additional data
```

```
     time_series          :Ttime_series;
     calc_conc            :boolean;
     dt50                 :double;
     lipid_normalisation  :boolean;
     lipid_content        :double;    //In percent wet weight
     error                :boolean;
     error_message        :string;
     iterations           :integer;
     likli                :double;
     ME                   :single;
     r2                   :single;
     slope                :single;
     intercept            :single;
     RMSD                 :single;
     //is toxicokinetics based on scaled length L/Lm?
     scaled_length        :boolean;
     volume_z             :boolean;
     L                    :double; //volumetric length
     Lm                   :double; //volumetric maximum length
    end;
    Tgutsarray=array of tguts;


    function simulate_GUTS(GUTS:tGUTS; h:integer;concentration:tdoublearray):TGUTS;


implementation



function initialize_GUTS(GUTS:TGUTS):TGUTS;
begin
  GUTS.cint:=0; GUTS.damage:=0; GUTS.h:=0;
  GUTS.fok.cint:=0; GUTS.sok.cint1:=0; GUTS.sok.cint2:=0;
  GUTS.Cum_Hazard:=0;
  GUTS.Survival:=1;
  result:=guts;
end;

function calculate_sdtk(Guts:Tguts):tguts;
var
dL:double;
a,b,c:double;
bc,ke:double;
oldweigth:double;
begin
a:=0.75;
b:=4.5;
c:=0.09;
dl:=((b+a)*c-c*guts.length)/24;
dl:=dl/guts.iterations;
guts.length:=guts.length+dl;
oldweigth:=guts.weight;
guts.weight:=exp(4.232+3.464*ln(guts.length))/1000;
guts.Cint:=guts.Cint*oldweigth/guts.weight;
BC:=power(10,-0.4297*math.log10(guts.weight)+2.8646);
ke:=-0.0045*sqrt(guts.weight)+0.0316;
```

```
guts.fok.kin:=(bc*ke)/(1-exp(-1*ke*24));
result:=guts;
end;


function toxicokinetic(GUTS:TGUTS):TGuts;
var
tk:boolean;
begin
  //Toxicokinetic model
     tK:=false;
     GUTS.error:=false;
     if GUTS.firstorderkinetic then
     begin
      if guts.sizedependtk then guts:=calculate_sdTK(guts);

      GUTS.fok.d_cint:=GUTS.fok.kin*guts.CW-GUTS.fok.kout*GUTS.fok.cint;
      GUTS.fok.cint:=GUTS.fok.cint+GUTS.fok.d_cint/GUTS.iterations;
      GUTS.Cint:=GUTS.fok.cint;
      tk:=true;
     end;
     if GUTS.secondorderkinetic and not tk then
     begin
      GUTS.sok.d_cint1:=(GUTS.sok.k11*guts.CW)+(GUTS.sok.k22*GUTS.sok.cint2)-(GUTS.sok.k12*
      guts.sok.cint1)-(guts.sok.k21*guts.sok.cint1);
      GUTS.sok.d_cint2:=(GUTS.sok.k21*GUTS.sok.cint1)-(GUTS.sok.k22*GUTS.sok.cint2);
      GUTS.sok.cint1:=GUTS.sok.cint1+GUTS.sok.d_cint1/GUTS.iterations;
      GUTS.sok.cint2:=GUTS.sok.cint2+GUTS.sok.d_cint2/GUTS.iterations;
      if guts.metabolite then guts.Cint:=guts.sok.cint2
      else GUTS.Cint:=GUTS.sok.cint1+GUTS.sok.cint2;
      tk:=true;
     end;
     if not tk then
     begin
      GUTS.error:=true;
      GUTS.error_message:='No toxicokinetic model selected!';
      exit;
     end;
     if GUTS.lipid_normalisation then GUTS.Cbiov:=GUTS.Cint*(100-GUTS.lipid_content)/100
                                 else GUTS.Cbiov:=GUTS.Cint;
     result:=GUTS;
end;

//*****************************************************************************
//**      Linking dose metrics to survival
function cal_it(GUTS:tGUTS):tGUTS;
begin //Linking dose metrics to survival by the assumption of individual tolerance
 guts.h:=guts.hb/guts.iterations;           //Calculate backgroundmortality of control
 guts.cum_hazard:=guts.Cum_Hazard+guts.h;   //Add the backgroundmortality to the cumulative
 hazard
 case guts.distribution of        //Calculate the individual tolerance distribution
  exponential:  guts.Survival:=exp(-1*guts.beta*math.max(guts.m,guts.m_old));
  loglogistic:  guts.Survival:=power(1+power(math.max(guts.m,guts.m_old)/guts.alpha,guts.beta
  ),-1);
  lognormal:    guts.Survival:=1-normalcdf(math.log10(math.max(guts.m,guts.m_old)),math.log10
  (guts.alpha),math.log10(guts.beta)/1.96);
  weibull:      guts.Survival:=1-WeibullCDF (math.max(guts.m,guts.m_old),guts.alpha,guts.beta
```

```
  );
 end;
 guts.Survival:=guts.Survival-(1-exp(-1*guts.Cum_Hazard)); //Reduce survival from individual
 tolerance distribution by the background mortality
 guts.m_old:=math.max(guts.m,guts.m_old);
 result:=guts;
end;

function cal_sd(GUTS:tGUTS):tGUTS;
begin  //Linking dose metrics to survival by the assumption of stochastic event
 if guts.volume_z=false then
 begin
  guts.h:=guts.kk*math.max(guts.M-guts.z,0)+guts.hb;
 end
 else
 //consider volume dependent z
 begin
  guts.h:=guts.kk*math.max(guts.M-(math.power(guts.L,3)*guts.z),0)+guts.hb;
 end;
 guts.h:=guts.h/guts.iterations;
 guts.cum_hazard:=guts.Cum_Hazard+guts.h;
 guts.Survival:=exp(-1*guts.Cum_Hazard);
 result:=guts;
end;

function survival_model(GUTS:tGUTS):tGUTS;
begin
  case guts.assumption of
    Without: result:=cal_sd(Guts);
    IT: result:=cal_it(guts);
    SD: result:=cal_sd(Guts);
  end;
end;
//*****************************************************************************
//** Calculate the dose metrics
function cal_ext_conc(GUTS:tGUTS):tGUTS;
begin
 guts.M:=guts.CW;
 result:=guts;
end;

function cal_int_conc(GUTS:tGUTS):tGUTS;
begin
 guts.M:=guts.Cint;
 result:=guts;
end;

function cal_scal_int_conc(GUTS:tGUTS):tGUTS;
var
d_CintSC:double;
begin
  if guts.scaled_length=false then
 begin
  if guts.scaled_uptake
  then d_CintSc:=guts.kin_scaled*guts.CW-guts.ke*guts.CintSc
  else  d_CintSc:=guts.ke*(guts.CW-guts.CintSc);
 end
```

```
 else
 begin {calculating scaled internal concentration based on scaled length}
  if guts.scaled_uptake
  then d_CintSc:=guts.kin_scaled*guts.CW-(guts.Lm/guts.L)*guts.ke*guts.CintSc
  else  d_CintSc:=(guts.Lm/guts.L)*guts.ke*(guts.CW-guts.CintSc);
 end;
  d_cintsc:=d_cintsc/guts.iterations;
  guts.CintSc:=guts.CintSc+d_CintSC;
  guts.M:=guts.CintSc;
  result:=guts;
end;

function cal_scal_dam(GUTS:tGUTS):tGUTS;
var
d_Damage:double;
begin
 guts:=toxicokinetic(guts);
 d_damage:=guts.kr*(guts.Cint-guts.DamageScaled);
 d_damage:=d_damage/guts.iterations;
 guts.DamageScaled:=guts.DamageScaled+d_damage;
 guts.M:=guts.DamageScaled;
 result:=guts;
end;

Function calculate_GUTS(GUTS:tGUTS):tGUTS;
begin
 case guts.dose_metric of
   Ext_Conc: guts:=cal_ext_conc(GUTS);
   Int_Conc: guts:=cal_int_conc(GUTS);
   Scal_Int_Conc: guts:=Cal_scal_int_conc(GUTS);
   Scal_Dam: guts:=cal_scal_dam(GUTS);
 end;
 result:=survival_model(GUTS);
end;


function set_dist_z(guts:tgutsarray;noi:integer):tgutsarray;
var
  I: Integer;
  dst:vector;
  sum_of_p:double;
begin
 //dst:=tdensemtxvec.Create;
 dst.size(2);
 sum_of_p:=0;
 for I := 0 to noi - 1 do
 begin
 if not guts[i].calibration then
 begin
   case guts[i].distribution of
     exponential: statrandom.RandomExponent(guts[i].beta,dst);
     loglogistic: statrandom.RandomLogistic(guts[i].alpha,guts[i].beta,dst);
     lognormal:   statrandom.RandomNormal(math.log10(guts[i].alpha),math.log10(guts[i].beta)/
     1.96,dst);
     weibull:     statrandom.RandomWeibull (guts[i].alpha,guts[i].beta,dst);
   end;
   guts[i].z:=dst.values[0];//max(dst[0],0);
```

```
 end
 else
 begin
   // Minimum probability =0; Maximum probability = 1
   // Screen from 0 to 1 and get values CDFinv
   // Calculate the probability of that value PDF
   case guts[i].distribution of
     exponential:
                  begin
                   guts[i].z:=expcdfinv(i/noi,guts[i].beta);
                   guts[i].pz:=expPDF(guts[i].z,guts[i].beta);
                  end;
     loglogistic:   //FUNKTIONIERT NOCH NICHT!
                  begin
                   guts[i].z:=expcdfinv(i/noi,guts[i].beta);
                   guts[i].pz:=expPDF(guts[i].z,guts[i].beta);
                  end;
     lognormal:
                  begin
                   guts[i].z:=normalcdfinv(math.log10(i/noi),math.log10(guts[i].alpha),math.
                   log10(guts[i].beta));
                   guts[i].pz:=normalPDF(guts[i].z,math.log10(guts[i].alpha),math.log10(guts[
                   i].beta));
                  end;
     weibull:
                  begin
                   guts[i].z:=weibullcdfinv(i/noi,guts[i].alpha,guts[i].beta);
                   guts[i].pz:=weibullPDF(guts[i].z,guts[i].alpha,guts[i].beta);
                  end;
   end;
   sum_of_p:=sum_of_p+guts[i].pz;
 end;
 end;
 for I := 0 to noi - 1 do guts[i].pz:=guts[i].pz/sum_of_p;
 result:=guts;
end;

function simulate_single_GUTS(GUTS:tGUTS; h:integer;concentration:tdoublearray):tGUTS;
var
i,t:integer;
survival_old:double;
t_s:Ttime_series;
dCW:double;
begin
  setlength(t_s,h+2);
  GUTS.survival:=1;
 for t := 0 to h do
  begin
    if concentration[t]>=0 then guts.CW:=concentration[t];
    t_s[t].timepoint:=t;
    t_s[t].concentration:=guts.CW;
    t_s[t].cint:=GUTS.Cint;
    t_s[t].damage:=GUTS.DamageScaled;
    t_s[t].M:=guts.M;
    t_s[t].hazard:=GUTS.Cum_Hazard;
    t_s[t].survival:=GUTS.survival;
    for i := 0 to GUTS.iterations-1 do
```

```
    begin
     //GUTS:=toxicokinetic(GUTS); //Toxicokinetic model
     //Lipid normalisation
     //if GUTS.lipid_normalisation then GUTS.Cbiov:=GUTS.Cint*(100-GUTS.lipid_content)/100
     //                           else GUTS.Cbiov:=GUTS.Cint;
     if guts.calc_conc then
     begin
      dCW:=guts.CW-(guts.CW*power(0.5,1/guts.dt50));
      guts.CW:=guts.CW-(dCW/guts.iterations);
     end;
     GUTS:=calculate_guts(GUTS);//Toxicodynamic model
    end;

  end;
  GUTS.time_series:=t_S;
  result:=guts;
end;
function simulate_GUTS(GUTS:tGUTS; h:integer;concentration:tdoublearray):tGUTS;
var
i,t:integer;
survival_old:double;
t_s:Ttime_series;
dCW:double;
pop:tgutsarray;
g: tguts;
begin
  if guts.assumption=without then
  begin
   setlength(pop,guts.noi);
   g:=guts;
   setlength(g.time_series,h+2);
   for t := 0 to trunc(h)+1 do g.time_series[t].survival:=0;

   for I := 0 to guts.noi - 1 do pop[i]:=guts;
   set_dist_z(pop,guts.noi);
   for I := 0 to guts.noi - 1 do
   begin
     pop[i]:=simulate_single_guts(pop[i],h,concentration);
   for t := 0 to trunc(h)+1 do
   begin
   if g.calibration then
    begin
     g.time_series[t].Survival:=g.time_series[t].Survival+pop[i].time_series[t].Survival*pop[
     i].pz;
    end
    else
    begin
     g.time_series[t].Survival:=g.time_series[t].Survival+pop[i].time_series[t].Survival/
     length(pop);
    end;
    g.time_series[t].timepoint:=pop[0].time_series[t].timepoint;
    g.time_series[t].concentration:=pop[0].time_series[t].concentration;
    g.time_series[t].cint:=pop[0].time_series[t].cint;
    g.time_series[t].damage:=pop[0].time_series[t].damage;
   end;
   end;
  end
```

```
  else g:=simulate_single_guts(guts,h,concentration);


  result:=g;
  setlength(pop,0);
end;

end.
```