Review: Numerical Methods for Differential Equations

ECE Generic and 2331

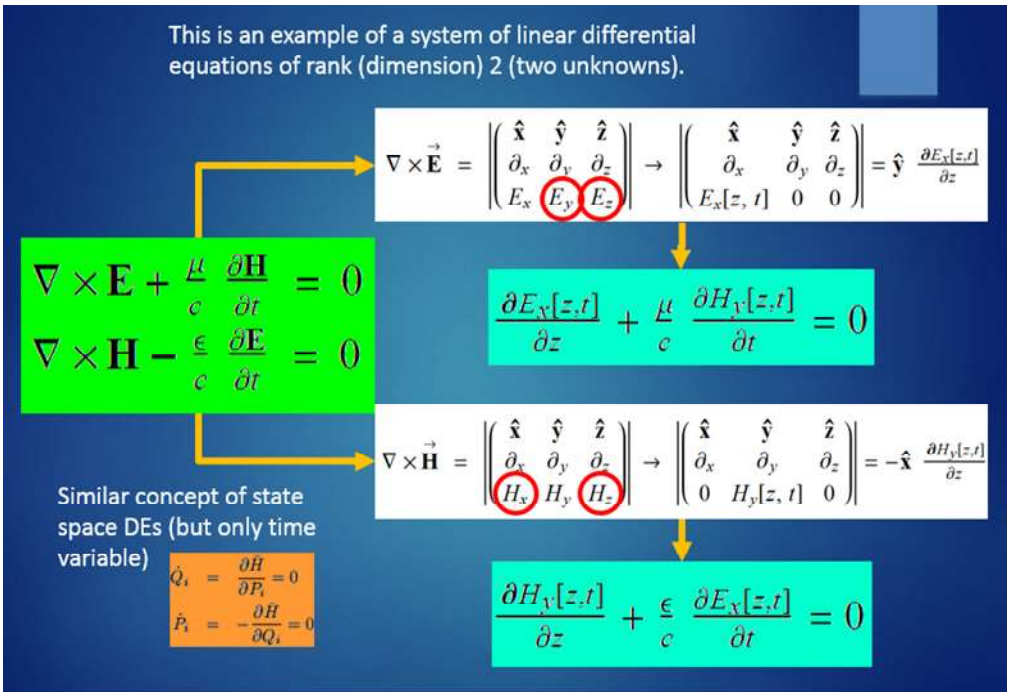
Han Q. Le(c)

Segment 2 - System and higher-order equations

---

1. Basic idea: multifunction and variable

More often than not, the real world problems involve many parameters. Instead of just a function y[x], we have u[x], v[x], etc... that are related to each other.



How do we apply techniques such as Euler's or R-K's to these problems? It is actually simpler than we think. For convenience, we treat the functions as a single entity, a vector function, like this:

$\frac{d\,y_1}{d\,x} = f_1(x, y_1, y_2, \ldots, y_n)$

$\frac{d y_2}{d x} = f_2(x, y_1, y_2, \ldots, y_n)$

...

$\frac{d y_n}{dx} = f_n(x, y_1, y_2, \ldots, y_n)$

or:

$$\frac{d}{dx}\begin{pmatrix} y_1 \\ y_2 \\ . \\ y_n \end{pmatrix} = \begin{pmatrix} f_1(x, y_1, y_2, \ldots, y_n) \\ f_2(x, y_1, y_2, \ldots, y_n) \\ . \\ f_n(x, y_1, y_2, \ldots, y_n) \end{pmatrix}$$

or:

$\frac{d}{dx}\mathbf{Y} = \mathbf{F}(x, \mathbf{Y})$

where **Y** is a vector and so is **F**(x,**Y**).

In some field, this is also known as a state-space model.

So far, we haven't done anything new, just a convenient notation.

But we can apply all previous methods to this problem

2. Euler's method for multifunction

2.1 Basic idea

Here is our equation : $\frac{d}{dx}\mathbf{Y} = \mathbf{F}(x, \mathbf{Y})$

For illustration, let's take **Y** to be of 3 – dimension.

From Taylor's series expansion, we recall that :

$$\begin{pmatrix} y_1(x_{i+1}) \\ y_2(x_{i+1}) \\ y_3(x_{i+1}) \end{pmatrix} \approx \begin{pmatrix} y_1(x_i) \\ y_2(x_i) \\ y_3(x_i) \end{pmatrix} + (x_{i+1} - x_i)\frac{d}{dx}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

So, we estimate :

$$\begin{pmatrix} y_1(x_{i+1}) \\ y_2(x_{i+1}) \\ y_3(x_{i+1}) \end{pmatrix} \approx \begin{pmatrix} y_1(x_i) \\ y_2(x_i) \\ y_3(x_i) \end{pmatrix} + h\begin{pmatrix} f_1(x_i; y_1(x_i), y_2(x_i), y_3(x_i)) \\ f_2(x_i; y_1(x_i), y_2(x_i), y_3(x_i)) \\ f_3(x_i; y_1(x_i), y_2(x_i), y_3(x_i)) \end{pmatrix}$$

We see that apart for writing everything 3 times, everything else is the same

2.2 Example

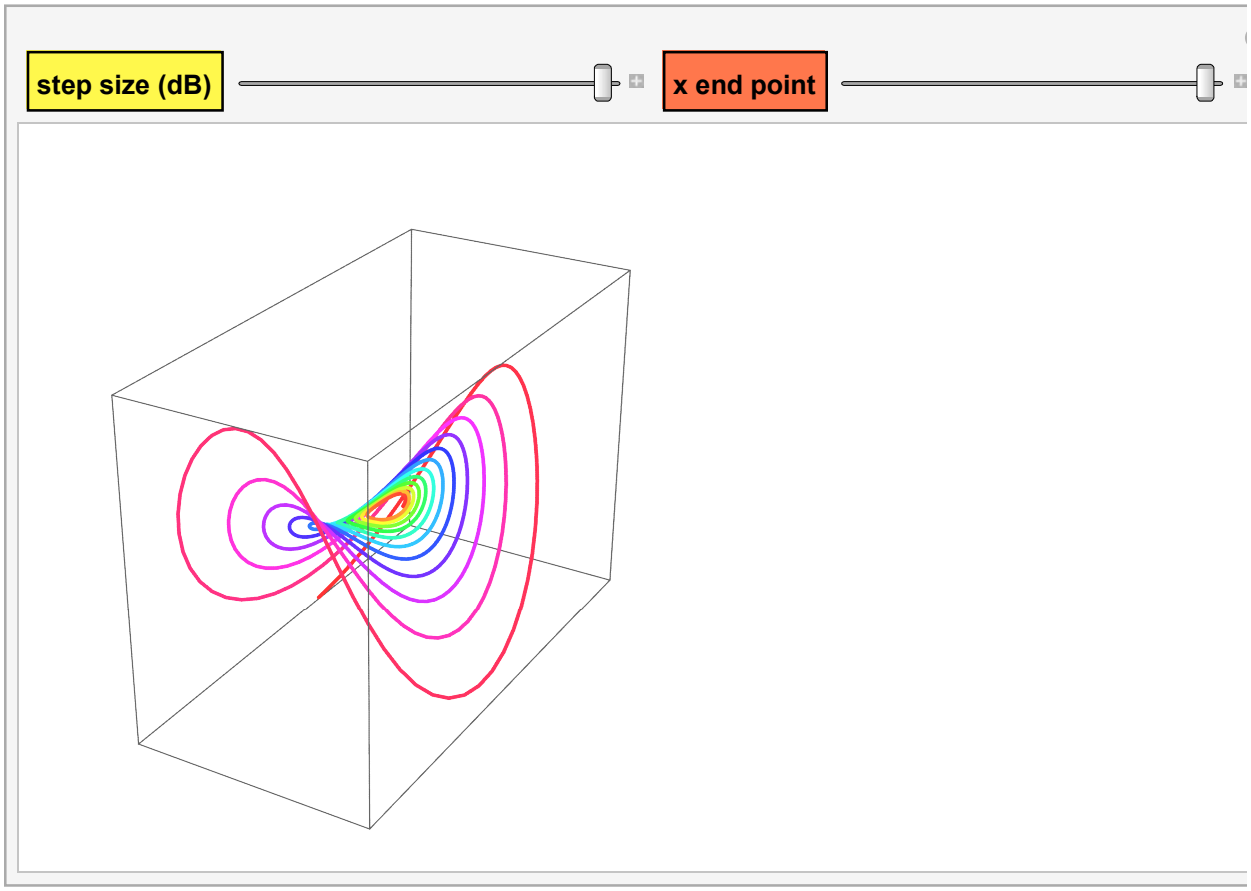
$$\frac{d}{dx}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} f_1(x, y_1, y_2, \ldots, y_n) \\ f_2(x, y_1, y_2, \ldots, y_n) \\ f_3(x, y_1, y_2, \ldots, y_n) \end{pmatrix} = \begin{pmatrix} -3(y_2 - y_3) \\ 15\, y_1 - y_1\, y_3 \\ 1.5\, y_1\, y_2 - y_3 \end{pmatrix}$$

If at $x = 0;$ $\begin{pmatrix} y_{1,1} \\ y_{2,1} \\ y_{3,1} \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix};$ then at $x = h = 0.1,$ we have an estimate :

$$\begin{pmatrix} y_{1\,\text{new}} \\ y_{2\,\text{new}} \\ y_{3\,\text{new}} \end{pmatrix} == \begin{pmatrix} y_{1\,\text{old}} \\ y_{2\,\text{old}} \\ y_{3\,\text{old}} \end{pmatrix} + h * \begin{pmatrix} -3\,(y_{2\,\text{old}} - y_{3\,\text{old}}) \\ 15\,y_{1\,\text{old}} - y_{1\,\text{old}}\,y_{3\,\text{old}} \\ 1.5\,y_{1\,\text{old}}\,y_{2\,\text{old}} - y_{3\,\text{old}} \end{pmatrix}$$

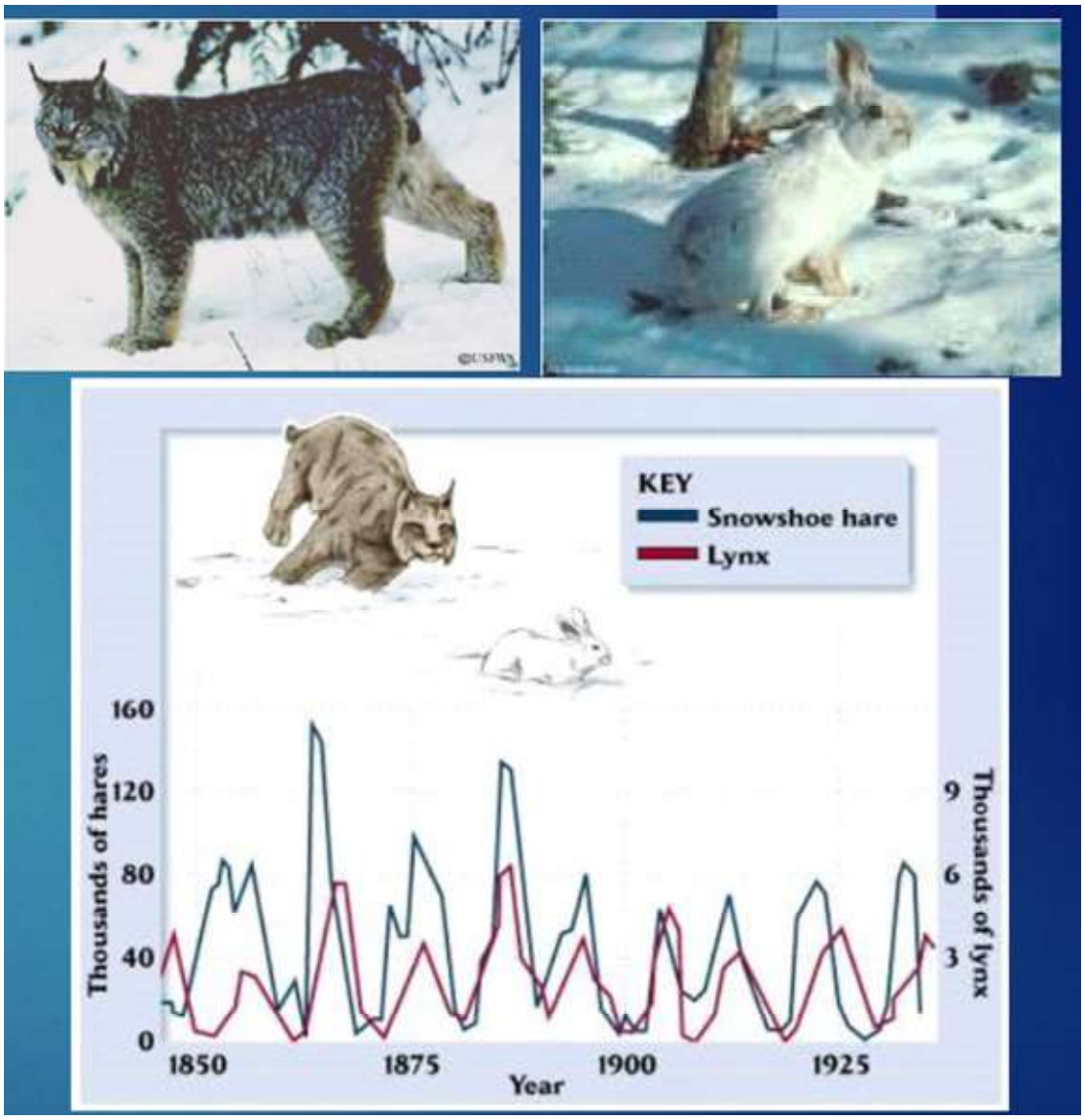
2.2.1 Algorithm

```
Manipulate[
 If[ ! TrueQ[{xendold, hdBold} == {xend, hdB}],
  h = Max[10^(0.1 hdB), 0.001 xend] ;
  If[ ! TrueQ[hold == h],
   np = Ceiling[xend / h];
   If[np ≤ 4, h = xend / 4.; np = Ceiling[xend / h];,];
   (y1 old)   = (1)
   (y2 old)     (1)
   (y3 old)     (0)  ; x = 0. ;
   {yvec, xarr} = Reap[For[i = 1, i ≤ np, {
         (y1 new)   = (y1 old)   + h * (  -3 (y2 old - y3 old)  )
         (y2 new)     (y2 old)     (  15 y1 old - y1 old y3 old )
         (y3 new)     (y3 old)     ( 1.5 y1 old y2 old - y3 old )  ;
         x = x + h ;
         Sow[{y1 new, y2 new, y3 new}, A]; Sow[x, B];
         (y1 old)   = (y1 new)
         (y2 old)     (y2 new)
         (y3 old)     (y3 new)  }, i++]][[2]];
   PrependTo[yvec, {1, 1, 0}];
   yvecgr = Table[{Hue[(i - 0)/Length[yvec] 1, 0.8],
      Line[{yvec[[i]], yvec[[i + 1]]}]}, {i, Length[yvec] - 1}];
   hold = h,];
  {xendold, hdBold} = {xend, hdB},];

 Graphics3D[{Thick, yvecgr}]

 , Grid[{
   {Control[{{hdB, -20,
       Button["step size (dB)", BaseStyle → {14, Bold, Black, FontFamily → "Arial"}
        , Background → Hue[0.16, 0.7, 1]]
      }, -40, -20, ImageSize → Medium}]
    , Control[
     {{xend, 10, Button["x end point", BaseStyle → {14, Bold, Black, FontFamily → "Arial"}
        , Background → Hue[0.04, 0.7, 1]]}, 5, 10., ImageSize → Medium}]
   }}]
 , Initialization :→ {
   {xendold, hdBold} = {-1., 0}

  }]
```



3. Example of Runge-Kutta

3.1 Problem set up



Now we see that besides the fact that we have to keep track of more than one functions, the rest is essentially the same as what we did before. Here we'll look at a 4th order R-K for a 2-function case:

$\frac{d}{dt}$Lynx = -0.9*Lynx + 0.5*Lynx*Hare

$\frac{d}{dt}$Hare = 1.1*Hare - 0.7*Lynx*Hare

3.2 Solution

We remember that the fourth order expression in $R-K$ method is :

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h;$$

where :

$k_1 = f(x_i, y_i);$

$$k_2 = f\left(x_i + \frac{1}{2}h,\ y_i + \frac{1}{2}k_1 h\right);$$

$$k_3 = f\left(x_i + \frac{1}{2}h,\ y_i + \frac{1}{2}k_2 h\right);$$

$$k_4 = f(x_i + h,\ y_i + k_3 h);$$

We set up now for a vector

$$\begin{pmatrix} y_{1\,\text{new}} \\ y_{2\,\text{new}} \end{pmatrix} = \begin{pmatrix} y_{1\,\text{old}} \\ y_{2\,\text{old}} \end{pmatrix} + \frac{1}{6}h \begin{pmatrix} k_{1,1} + 2k_{1,2} + 2k_{1,3} + k_{1,4} \\ k_{2,1} + 2k_{2,2} + 2k_{2,3} + k_{2,4} \end{pmatrix};$$

where :

$$\begin{pmatrix} k_{1,1} \\ k_{1,2} \\ k_{1,3} \\ k_{1,4} \end{pmatrix} = \begin{pmatrix} f_1(x_i,\ y_{1,i},\ y_{2,i}) \\ f_1\left(x_i + \frac{1}{2}h,\ y_{1,i} + \frac{1}{2}k_{1,1}h,\ y_{2,i} + \frac{1}{2}k_{2,1}h\right) \\ f_1\left(x_i + \frac{1}{2}h,\ y_{1,i} + \frac{1}{2}k_{1,2}h,\ y_{2,i} + \frac{1}{2}k_{2,2}h\right) \\ f_1(x_i + h,\ y_{1,i} + k_{1,3}h,\ y_{2,i} + k_{2,3}h) \end{pmatrix};$$ and similarly :

$$\begin{pmatrix} k_{2,1} \\ k_{2,2} \\ k_{2,3} \\ k_{2,4} \end{pmatrix} = \begin{pmatrix} f_2(x_i,\ y_{1,i},\ y_{2,i}) \\ f_2\left(x_i + \frac{1}{2}h,\ y_{1,i} + \frac{1}{2}k_{1,1}h,\ y_{2,i} + \frac{1}{2}k_{2,1}h\right) \\ f_2\left(x_i + \frac{1}{2}h,\ y_{1,i} + \frac{1}{2}k_{1,2}h,\ y_{2,i} + \frac{1}{2}k_{2,2}h\right) \\ f_2(x_i + h,\ y_{1,i} + k_{1,3}h,\ y_{2,i} + k_{2,3}h) \end{pmatrix}$$

Now, we see how convenient it is to deal everything as a vector :

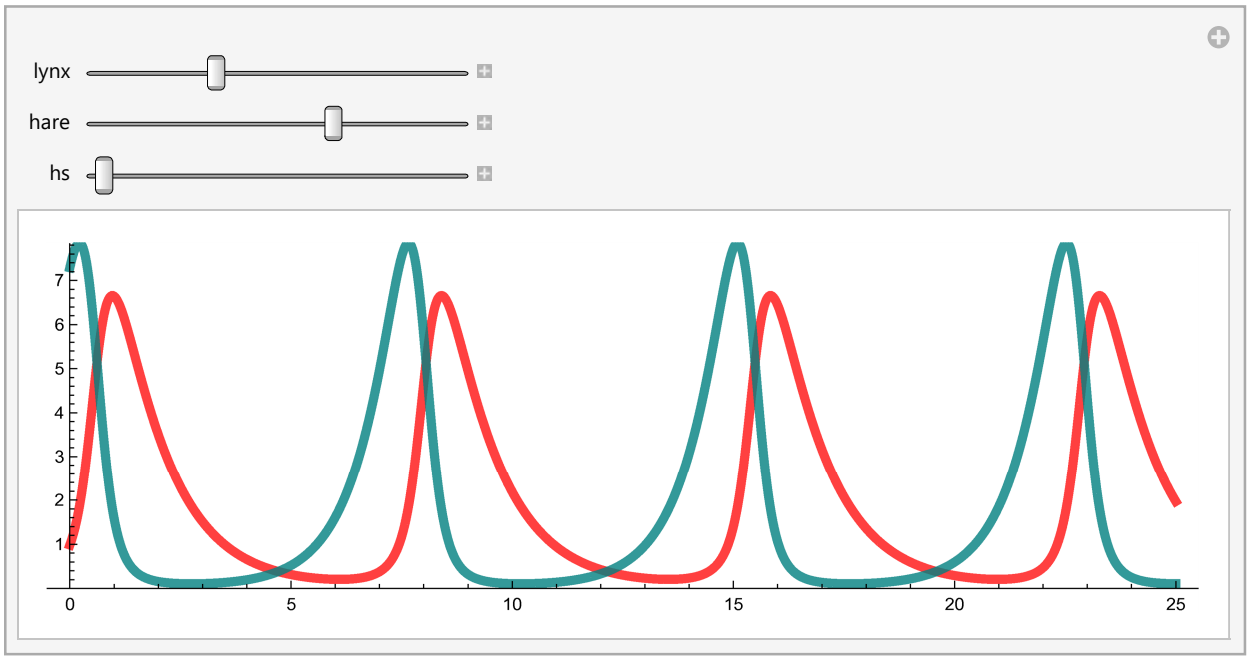
$$\begin{pmatrix} k_{1,1} & k_{2,1} \\ k_{1,2} & k_{2,2} \\ k_{1,3} & k_{2,3} \\ k_{1,4} & k_{2,4} \end{pmatrix} = \begin{pmatrix} f_1(x;\ Y_i) & f_2(x;\ Y_i) \\ f_1\left(x + \frac{1}{2}h;\ Y_i + \frac{1}{2}K_1 h\right) & f_2\left(x + \frac{1}{2}h;\ Y_i + \frac{1}{2}K_1 h\right) \\ f_1\left(x + \frac{1}{2}h;\ Y_i + \frac{1}{2}K_2 h\right) & f_2\left(x + \frac{1}{2}h;\ Y_i + \frac{1}{2}K_1 h\right) \\ f_1(x + h;\ Y_i + K_3 h) & f_2\left(x + \frac{1}{2}h;\ Y_i + \frac{1}{2}K_1 h\right) \end{pmatrix}$$

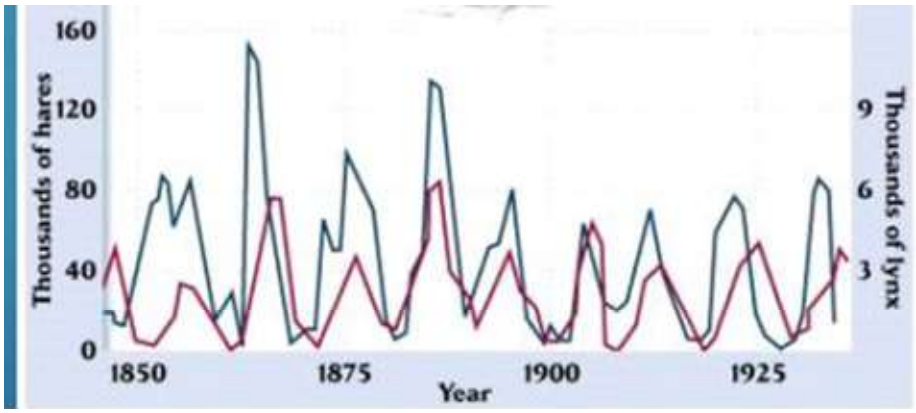
3.2.1 Simple code

```
Manipulate[
 If[! TrueQ[{lynxold, hareold, hsold} == {lynx, hare, hs}],
  (y1 old
   y2 old) = (lynx
              hare);
  x = 0. ; np = Ceiling[25. / hs];
  np = Min[1000, np]; h = 25 / np;
lynxinit = {x, y1 old}; hareinit = {x, y2 old};
Yold = (y1 old
        y2 old) ;
{lynxPop, harePop} = Reap[For[i = 1, i ≤ np, i++, {
```

```
        (k11)   = Flh[x, Yold] ;
        (k21)

        (k12)   = Flh[x + h/2, Yold + (k11)  * h/2] ;
        (k22)             (k21)

    (k13)   = Flh[x + h/2, Yold + (k12)  * h/2] ;
    (k23)             (k22)

        (k14)   = Flh[x + h, Yold + (k13)  * h] ;
        (k24)             (k23)

    (y1 new)   (y1 old)   h   (k11 + 2 k12 + 2 k13  + k14 )
    (y2 new) = (y2 old) + - * (k21 + 2 k22 + 2 k23  + k24 ) ;
                          6

    x = x + h ;
        Sow[{ x , y1 new }, A] ;
        Sow[{ x , y2 new }, B] ;
          (y1 old)   (y1 new)          (y1 old)
          (y2 old) = (y2 new) ; Yold = (y2 old)
            }]][[2]];
  PrependTo[lynxPop, lynxinit]; PrependTo[harePop, hareinit];
  maxpop = Max[{Transpose[lynxPop][[2]], Transpose[harePop][[2]]}];
  LHplot = ListPlot[{lynxPop, harePop}, Joined -> True,
PlotStyle -> {{Hue[0., 1., 1, 0.75], Thickness[0.008]},
                {Hue[0.5, 1, 0.5, 0.8], Thickness[0.008]}}
    , PlotRange → {0, maxpop}
, ImageSize → 600, AspectRatio → 0.3] ;
  (* {t,xarr}=Transpose[theta]; *)
  {lynxold, hareold, hsold} = {lynx, hare, hs}
  ,];

 LHplot

 , {{lynx, 1}, 0.5, 2}, {{hare, 5}, 2, 10}, {{hs, 0.1}, 0.01, 2}
 , Initialization :⇾ {
   Flh[x_, Y_] := ( (-0.9 * Y[[1]] + 0.5 * Y[[1]] * Y[[2]]) [[1]] )
                  ( (1.3 * Y[[2]] - 0.7 * Y[[1]] * Y[[2]]) [[1]] ) ;
    ;}]
```

3.2.2 Output demo



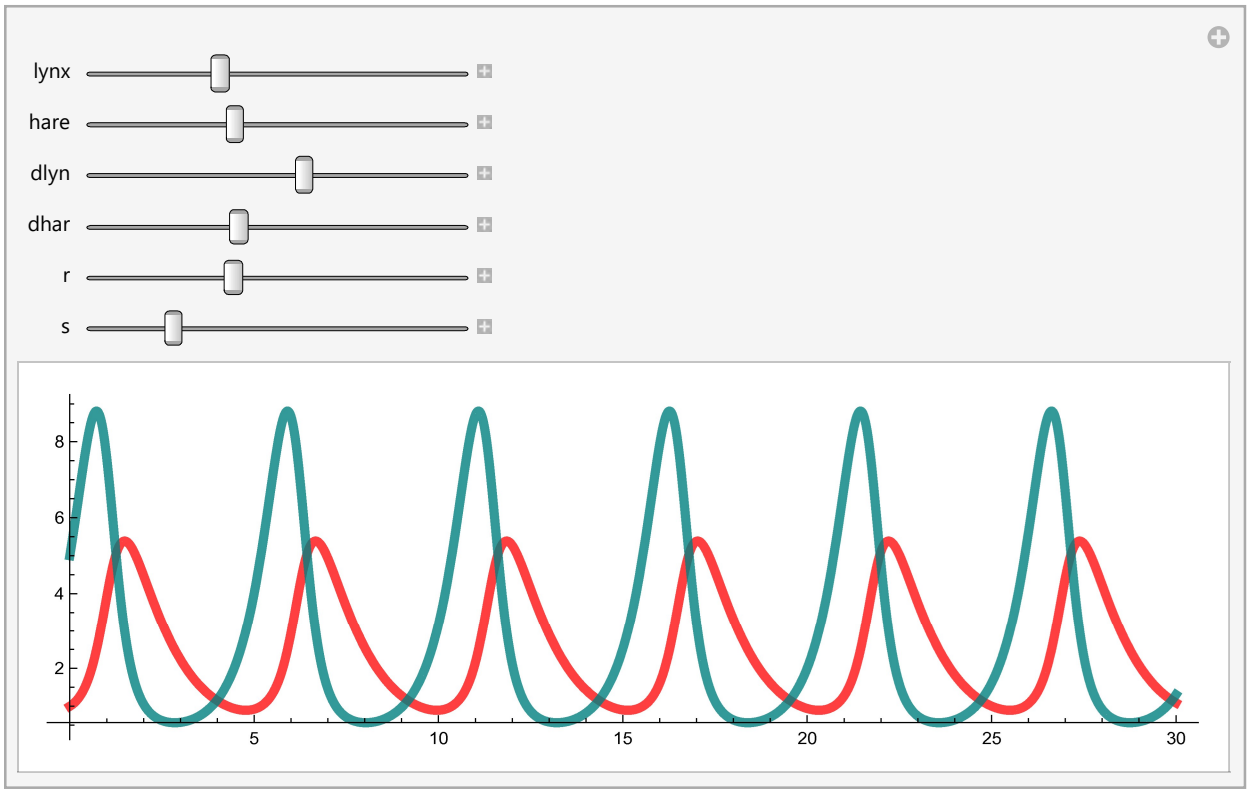


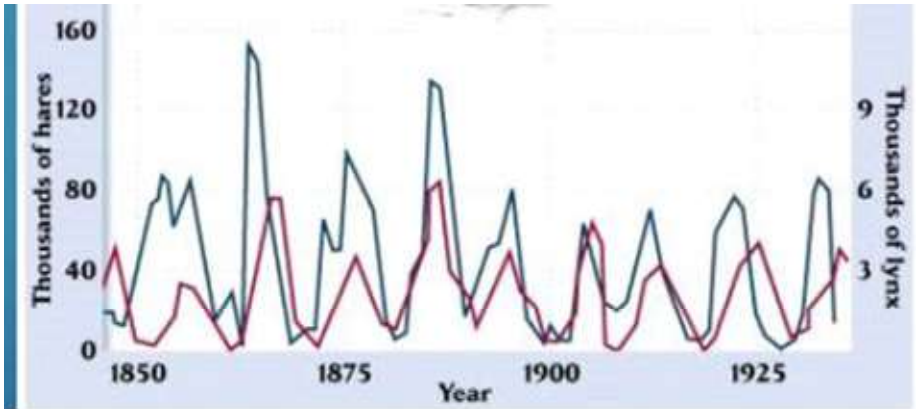
This is the famous lynx/hare population behavior: we start with a high population of hares (teal) which is an abundant source of food for the lynx (red) with a rapid population growth. But as the lynx population grows large, they deplete the hare population, which declines precipitously, leading to lynx starvation. The lynx population declines, giving the hares a chance to rebuild their population and the cycle repeats itself.

3.3 Using NDSolve

In general, we don't have to write own codes of common algorithm such as R-K because numerous software packgages have available to use. Below is an example with NDSolve:

```
Manipulate[
 popParm = {{dlyn, r}, {s, dhar}};
 popinit = {lynx, hare};
 If[! TrueQ[{popParmold, popold} == {popParm, popinit}]
  ,
  solLH = NDSolve[{{yL'[t], yH'[t]} == fLH[t, {yL[t], yH[t]}, popParm],
     {yL[0.], yH[0.]} == popinit}, {yL, yH}, {t, 0, 30}];
  lhPlot = Plot[Release[{yL[t], yH[t]} /. solLH], {t, 0., 30}
    , PlotRange → All
    , PlotStyle → {{Hue[0., 1., 1, 0.75], Thickness[0.008]},
      {Hue[0.5, 1, 0.5, 0.8], Thickness[0.008]}}
    , ImageSize → 600, AspectRatio → 0.3] ;
  {popParmold, popold} = {popParm, popinit}
  ,];
 lhPlot
 , {{lynx, 1}, 0.5, 2}, {{hare, 5}, 2, 10}
 , {{dlyn, -0.9}, -2, -0.1}, {{dhar, 2.}, 0.1, 5}
 , {{r, 0.3}, 0.001, 0.8}, {{s, -0.8}, -1, 0.001}
 , Initialization :→ {
   fLH[x_, Y_, parm_] := {parm[[1, 1]] * Y[[1]] + parm[[1, 2]] * Y[[1]] * Y[[2]]
     , parm[[2, 2]] * Y[[2]] + parm[[2, 1]] * Y[[1]] * Y[[2]]} ;
  }]
```





4. Higher order differential equation

4.1 Basic concept

We see that solving first-order DE (involving only the first-order derivative) by numerical method is straight forward. So if we encounter a higher order equation, like this:

f(x, y,...) $\frac{d^n y}{d x^n}$ + g(x, y,..)$\frac{d^{n-1} y}{d x^{n-1}}$ + ... =0

we should convert if possible to first order.

Example, let's look at the Galileo's pendulum problem:

$\frac{d^2 \theta}{d t^2}+\frac{g}{l}$ sin(θ)=0:

which is second order. So we define an intermediate function:

Ω(t) = $\frac{d \theta}{d t}$; and now we have a systems of two equations:

$\frac{d \Omega}{d t}$= - $\frac{g}{l}$ sin(θ) ;

$\frac{d \theta}{d t}$ =Ω(t)

4.2 Example: R-K method to solve the pendulum problem

We can use the same algorithm developed for the lynx-hare problem above for this case. We just need to make a different function

4.2.1 Pendulum simple code

```
Manipulate[
 If[! TrueQ[{θold, hsold} == {θinit, hs}],
  (y1 old
   y2 old) = (0.
              θinit); x = 0. ; np = Ceiling[25. / hs];
  np = Min[1000, np]; h = 25 / np;
omegainit = {x, y1 old}; thetainit = {x, y2 old};
```

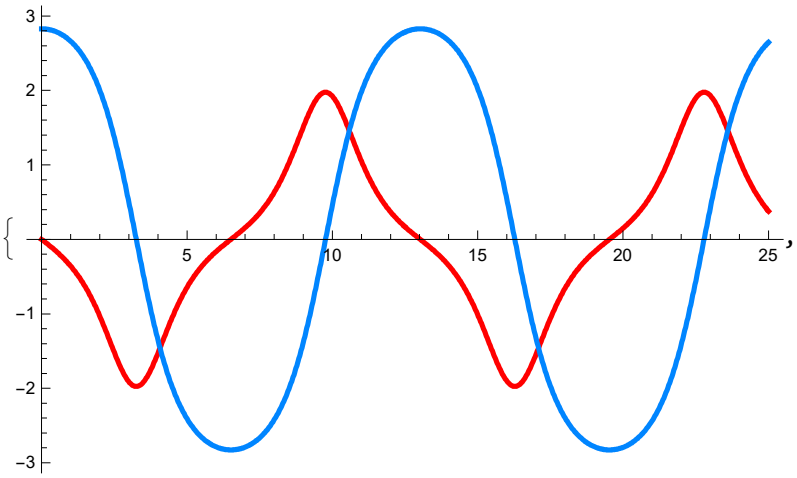
```
Y_old = ( y1 old
          y2 old ) ;
{omega, theta} = Reap[For[i = 1, i ≤ np, i++, {
        ( k11
          k21 ) = F[x, Y_old] ;
        ( k12
          k22 ) = F[x + h/2, Y_old + ( k11
                                       k21 ) * h/2] ;
    ( k13
      k23 ) = F[x + h/2, Y_old + ( k12
                                   k22 ) * h/2] ;
        ( k14
          k24 ) = F[x + h, Y_old + ( k13
                                     k23 ) * h] ;
    ( y1 new
      y2 new ) = ( y1 old
                   y2 old ) + h/6 * ( k11 + 2 k12 + 2 k13 + k14
                                      k21 + 2 k22 + 2 k23 + k24 ) ;
    x = x + h ;
        Sow[{ x , y1 new }, A] ;
        Sow[{ x , y2 new }, B] ;
          ( y1 old
            y2 old ) = ( y1 new
                         y2 new ) ; Y_old = ( y1 old
                                              y2 old )
           }]][[2]];
  PrependTo[omega, omegainit]; PrependTo[theta, thetainit];
  pendgr = ListPlot[{omega, theta}, Joined -> True,
PlotStyle -> {{Hue[0., 1, 1], Thickness[0.007]},
                {Hue[0.58, 1, 1], Thickness[0.007]}}
, ImageSize → 400] ;
  {t, xarr} = Transpose[theta];
  {θold, hsold} = {θinit, hs}
  ,];
 {pendgr,
  Manipulate[
   Graphics[{Hue[0.5, 1., 1.], Thickness[0.01],
     Line[{{0., 0.}, {Sin[xarr[[i]] ], -Cos[xarr[[i]] ]}}]
     , White, PointSize[0.02], Point[{0., 0.}],
     Red, Disk[{Sin[xarr[[i]] ], -Cos[xarr[[i]] ]}, 0.1]
     , Inset[NumberForm[(i - 1.) * h, 2], ImageScaled[{0.9, 0.9}]
      , BaseStyle → {24, Yellow}]}
    , PlotRange → {{-1.25, 1.25}, {-1.25, 1.25}}
    , Background → Black]
   , {i, 1, Length[xarr], 1, ControlType → Animator, DefaultDuration → 8
    , AnimationRunning → False}]
 }

 , {{θinit, Pi / 2}, 0.01, 0.9 Pi}, {{hs, 0.01}, 0.005, 0.5}
 , Initialization :↦ {
```

```
F[t_, Y_List] := {-Sin[Y[[2]]][[1]]
                   Y[[1]][[1]]};
```

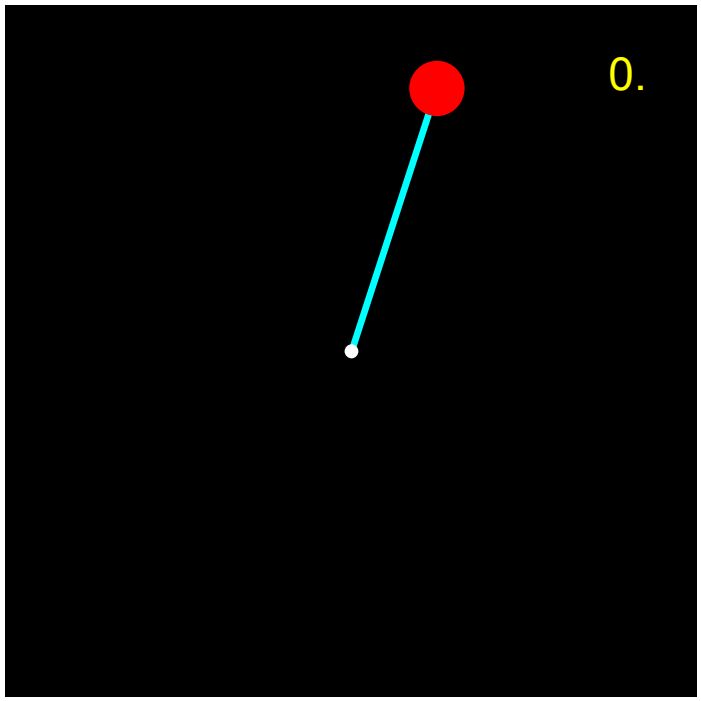
4.2.2 output

θinit

hs



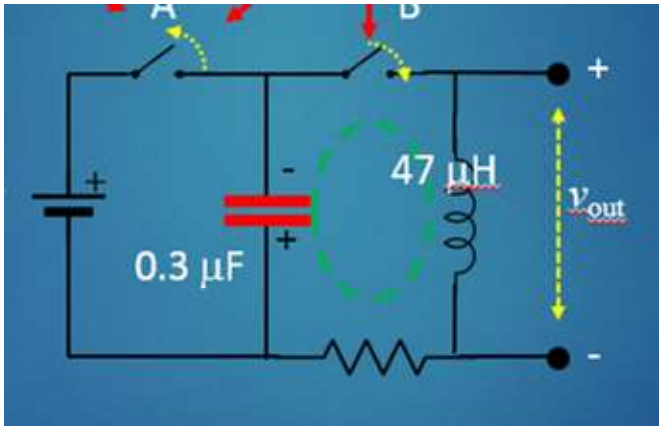
i



4.3 Pendulum with NDSolve

4.4 Example: R-K method for simple RLC circuit (damped oscillation)

Below is a circuit



With Laplace transform, we can determine the behavior based on the poles of the transfer function.

Underdamped, complex roots near Im axis, highly oscillatory response

Decompose Laplace Transform Transfer Function

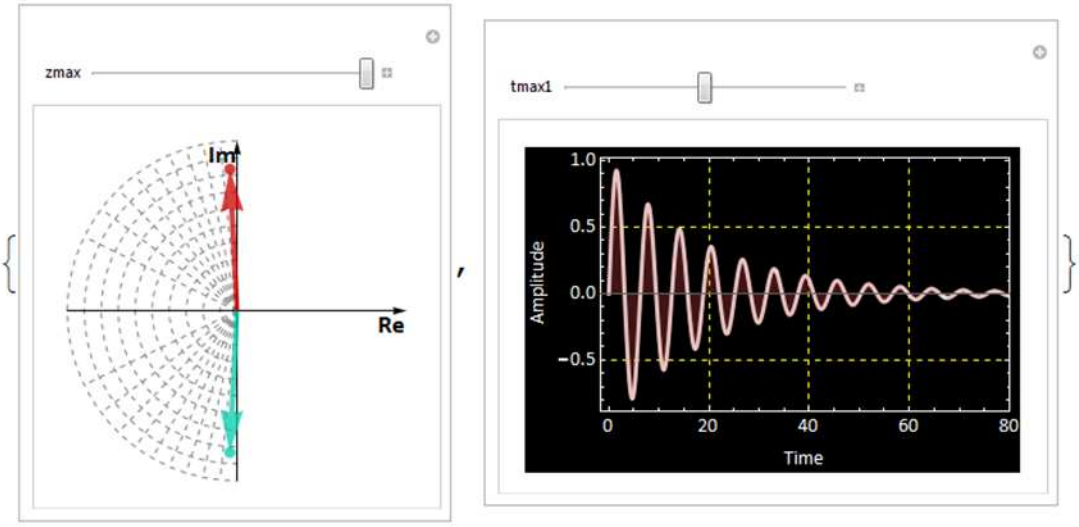
FLoating/Integer option→ FLOATING DONE
(can be slow for n>=3)

$$\frac{1}{1.+0.102\,s+s^2} = -\frac{0.+0.5\,i}{(0.051-1.\,i)+1.\,s} + \frac{0.+0.5\,i}{(0.051+1.\,i)+1.\,s}$$

Response function

$$(0.+0.5\,i)\,e^{(-0.051-1.\,i)\,t} - (0.+0.5\,i)\,e^{(-0.051+1.\,i)\,t}$$

Roots and Decomposed Response Functions



Overdamped, real negative roots, rapid exponential decay response

Decompose Laplace Transform Transfer Function

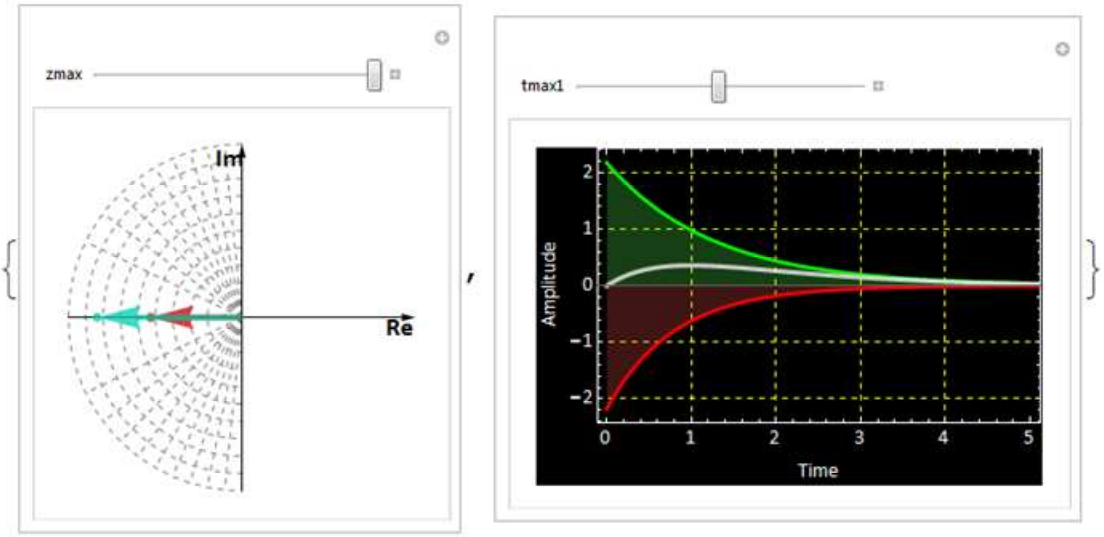
FLoating/Integer option→ FLOATING DONE
(can be slow for n>=3)

$$\frac{1}{1.+2.052\,s+s^2} = \frac{2.2}{0.8+1.\,s} - \frac{2.2}{1.3+1.\,s}$$

Response function

$$-2.2\,e^{-1.3\,t} + 2.2\,e^{-0.8\,t}$$

Roots and Decomposed Response Functions



Of course we know the solution analytically, but we use this example to show how to apply Runge-Kutta and see how it works.

Let the general equation be:

$\frac{d^2 y}{dt^2} + 2\,\zeta\,\omega_n \frac{dy}{dt} + \omega_n{}^2\, y = 0$:

which is second order. So we define an intermediate function:

$y_1(t) = \frac{dy}{dt}$; and now we have a systems of two equations:

$\frac{dy_1}{dt} = -2\,\zeta\,\omega_n\, y_1 - \omega_n{}^2\, y_0$

$\frac{dy_0}{dt} = y_1(t)$

We can use the same algorithm developed for the lynx-hare problem above for this case. We just need to make a different vector function

```
Manipulate[
 If[! TrueQ[{ωnold, ζold, xendold, hsold} == {ωn, ζ, xend, hs}],
  h = Max[hs, 0.001 xend] ;
  np = Ceiling[xend / h];
  If[np ≤ 4, h = xend / 4.; np = Ceiling[xend / h];,];
  (y1 old
   y2 old) = (0.
              1); x = 0. ;
```

```
volt0init = {x, y1 old};
volt1init = {x, y2 old};
Yold = ( y1 old
       y2 old ) ;
  {volt0, volt1} = Reap[For[i = 1, i ≤ np, i++, {
       (k11
        k21) = Fosc[Yold, {ωn, ζ}] ;
       (k12
        k22) = Fosc[Yold + (k11
                            k21) * h/2, {ωn, ζ}] ;
       (k13
        k23) = Fosc[Yold + (k12
                            k22) * h/2, {ωn, ζ}] ;
       (k14
        k24) = Fosc[Yold + (k13
                            k23) * h, {ωn, ζ}] ;
       (y1 new
        y2 new) = (y1 old
                   y2 old) + h/6 * (k11 + 2 k12 + 2 k13 + k14
                                    k21 + 2 k22 + 2 k23 + k24) ;
       x = x + h ;
       Sow[{x , y1 new}, V1]; Sow[{x , y2 new}, V2] ;
       (y1 old
        y2 old) = (y1 new
                   y2 new) ; Yold = (y1 old
                                     y2 old)
          }]][[2]];
  PrependTo[volt0, volt0init]; PrependTo[volt1, volt1init];

  {ωnold, ζold, xendold, hsold} = {ωn, ζ, xend, hs}
  ,];
 ListPlot[{volt0, volt1}, Joined → True
, PlotStyle -> {{RGBColor[1., 0., 0], Thickness[0.007]},
                {RGBColor[0., 0., 1], Thickness[0.007]}}
  , Frame → True, GridLines → Automatic
  , FrameLabel → {"time", "voltage"}
  , LabelStyle → {16, Bold, Black, FontFamily → "Arial"}
, ImageSize → {400., 300}
, PlotRange → {-1.1, 1.1}]
 , Grid[{{Control[
     {{ωn, 1., Button["nat. frequency", BaseStyle → {14, Bold, FontFamily → "Calibri"},
        Background → Hue[0.8, 0.4, 1]]}, 0.5, 5, 0.1}]
    , Control[{{xend, 20, Button["end time", BaseStyle → {14, Bold,
          FontFamily → "Calibri"}, Background → Hue[0.25, 0.8, 1]]}, 1, 40, 0.025}]}
   , {Control[{{ζ, 0.2, Button["damping coef.", BaseStyle → {14, Bold,
          FontFamily → "Calibri"}, Background → Hue[0.07, 0.8, 1]]}, 0.001, 2., 0.025}]
    , Control[{{hs, 0.1, Button["step size", BaseStyle → {14, Bold,
          FontFamily → "Calibri"}, Background → Hue[0.5, 0.4, 1]]}, 0.01, 0.5}]
   }}]
 , Initialization :⇒ {
```

```
Fosc[Y_, {ωn_, ζ_}] := ( Y[[2, 1]]
                         -ωn (ωn Y[[1, 1]] + 2 ζ Y[[2, 1]]) ) ;
```

}]