I am not liable for any thing

Log_dif_info(inflection points,and solutions given r,k and ρo in the form dρ/dt=rρ(1-ρ/k) and ρ(0) = ρ0)

```
Define LibPub log_diff_info(r,k,ρ0,t)=
Func
:©log_diff_info(r,k,ρ0,var) in the form *((dρ)/(dt))=rρ(1-((ρ)/(k))) and ρ(0)=ρ0
:Local solution,maxρ,maxt
:solution:=((ρzero*capρ)/(ρzero+(capρ-ρzero)*^(-rate*t)))
:Disp "ρ(t)=",solution
:solution:=solution|capρ=k and ρzero=ρ0 and rate=r
:Disp "ρ(t)=",solution
:maxρ:=((k)/(2))
:Disp "max rate of change:ρ=",maxρ
:maxt:=zeros(solution-maxρ,t)
:Disp "max rate of change:t=",maxt
:Return solution
:EndFunc
```

$$log\_diff\_info\left(\frac{25}{1000},1000,20,t\right)$$

$$p(t)=\frac{cap\cdot pzero\cdot e^{rate\cdot t}}{pzero\cdot e^{rate\cdot t}+cap-pzero}$$

$$p(t)=\frac{1000\cdot e^{\frac{t}{40}}}{e^{\frac{t}{40}}+49}$$

$$\text{max rate of change:p}=500$$

$$\text{max rate of change:t}=\{80\cdot\ln(7)\}$$

$$\frac{1000\cdot e^{\frac{t}{40}}}{e^{\frac{t}{40}}+49}$$

vectproj(infomation about vector project)

```
Define  LibPub vectproj(a,b)=
Func
:Disp "a.b/b.b:",((dotP(a,b))/(dotP(b,b)))
:Disp "scaler resolute:a.b hat:",sresolute(a,b)
:Disp "vectResolute:",vresolute(a,b)
:Disp "perpendicular:a-VectorResolute:",wresolute(a,b)
:Return 0
:EndFunc
```

$vectproj([1 \quad 2 \quad 3],[3 \quad -4 \quad -1])$

a.b/b.b: $\frac{-4}{13}$

a.unitv(b)

scaler resolute:a.b hat: $\frac{-4\cdot\sqrt{26}}{13}$

vectResolute: $\begin{bmatrix} \frac{-12}{13} & \frac{16}{13} & \frac{4}{13} \end{bmatrix}$

a−((a.b)/(b.b))Xb

perpendicular:a−VectorResolute: $\begin{bmatrix} \frac{25}{13} & \frac{10}{13} & \frac{35}{13} \end{bmatrix}$

0

sresolute(computes scaler resolute of a in direction b)

```
Define LibPub sresolute(a,b)=
Func
:Disp "a.unitv(b)"
:Return dotP(a,unitV(b))
:EndFunc
```

$sresolute([1 \quad 2 \quad 3],[-3 \quad 4 \quad 2])$

a.unitv(b)

$\frac{11\cdot\sqrt{29}}{29}$

vresolute(computes vector resolute of a in direction b)

```
Define  LibPub vresolute(a,b)=
Func
:Return ((dotP(a,b))/(dotP(b,b)))*b
:EndFunc
```

$vresolute([1 \quad 2 \quad 3],[-3 \quad -4 \quad -1])$ $\begin{bmatrix} \frac{21}{13} & \frac{28}{13} & \frac{7}{13} \end{bmatrix}$

wresolute(perpendictular component of a)

```
Define  LibPub wresolute(a,b)=
Func
:Disp "a-((a.b)/(b.b))Xb"
:Return a-vresolute(a,b)
:EndFunc
```

$wresolute([1 \quad 2 \quad 3],[3 \quad -4 \quad -1])$

$a-((a.b)/(b.b))Xb$

$$\begin{bmatrix} \frac{25}{13} & \frac{10}{13} & \frac{35}{13} \end{bmatrix}$$

Finds angle between vector a and b

```
Define  LibPub vectangle(a,b)=
Func
:Disp "(a.b)/(|a||b|):",((dotP(a,b))/(norm(a)*norm(b)))
:Return arccos(((dotP(a,b))/(norm(a)*norm(b))))
:EndFunc
```

$vectangle([1 \quad 0],[\sqrt{2} \quad \sqrt{2}])$

$(a.b)/(|a||b|): \frac{\sqrt{2}}{2}$

$$\frac{\pi}{4}$$

vectore_equ(breakes down vector equation into constants and anything else)

```
Define  LibPub vectore_equ(r,t)=
Func
:Local d,a
:d:=∫((r,t),t)
:a:=r-d
:Disp "a:",a
:Disp "d:",d
:Return d
:EndFunc
```

$vectore\_equ([1+2\cdot t \quad c+7\cdot t \quad 1],t)$

$a:[1 \quad c \quad 1]$
$d:[2\cdot t \quad 7\cdot t \quad 0]$

$[2\cdot t \quad 7\cdot t \quad 0]$

vectore_dir(vector d,parral vector to line)
*assumes r is a line

```
Define  LibPub vectore_dir(r,t)=
Func
:Local d,a
:d:=∫((r,t),t)
```

```
:a:=r-d
:Return d/t
:EndFunc
```

$vectore\_dir\left(\begin{bmatrix}1 & 2\cdot t & c+7\cdot t\end{bmatrix},t\right)$ $\begin{bmatrix}0 & 2 & 7\end{bmatrix}$

vector_d2l(distace from line r to point ρ)
*sometimes gives multiple solutions(choses the non zero solution)

```
Define LibPub vector_d2l(r,ρ,t)=
Func
: Local ρq,shortt
: ρq:=r-ρ
: Local equ,d1
: d1:=vectore_dir(r,t)
: equ:=dotP(ρq,d1)
: Disp "((x,y,z..)-ρ).d1=",equ,"=0"
: shortt:=zeros(equ,t)
: Disp "t:",shortt
: Local distlist,minsh
:  minsh:=norm(ρq)|t=shortt[1]
:  Local n,i
:  n:=count(shortt)
:  For i,1,n
:    distlist[i]:=norm(ρq)|t=shortt[i]
:    If minsh>distlist[i] Then
:      minsh:=distlist[i]
:      EndIf
:      EndFor
:      Disp "dist:",distlist
:      Return minsh
:EndFunc
```

$vector\_d2l\left(\begin{bmatrix}1-t & 2-3\cdot t & 2\end{bmatrix},\begin{bmatrix}1 & 3 & 2\end{bmatrix},t\right)$

$((x,y,z..)-p).d1=10\cdot t+3=0$

$t:\left\{\frac{-3}{10}\right\}$

$dist:\left\{\frac{\sqrt{10}}{10}\right\}$

$\frac{\sqrt{10}}{10}$

VectPlane(vector plane from 3 position vectors)

*assumes 3d,works in 2d probably not for higher dementions

```
Define LibPub vectplane(p1,p2,p3)=
Func
:Local sg1,sg2,normal
:sg1:=p1-p2
:sg2:=p3-p2
:Disp "p2->p1",sg1
:Disp "p2->p3",sg2
:normal:=crossP(sg1,sg2)
:Disp "(p2->p1)X(p2->p3)",normal
:Disp normal,".","(i,j,k...)=",dotP(p1,normal)
:Return dotP(p1,normal)
:EndFunc
```

vectplane([0 1 1],[2 1 0],[-2 0 3])

```
                                  p2->p1[-2  0  1]
                                  p2->p3[-4 -1  3]
                                  (p2->p1)X(p2->p3)[1  2  2]
                                  [1  2  2] . (i,j,k...)=4
                                                          4
```

1x+2y+2z=4 equvilant

distance_pl2p(distance from plane with normal n to point p)
an as in plane equation dot(a,n) = dot(r,n)

```
Define LibPub distance_pl2p(p,n,an)=
Func
:Local fin
:fin:=dotP(n,-p)
:Disp "((x,y,z..)-p).n hat"
:Return ((an+fin)/(norm(n)))
:EndFunc
```

distance_pl2p([1 -4 -3],[2 -3 6],-1)

```
                                  ((x,y,z..)-p).n hat
                                                    3
                                                    -
                                                    7
```

vector_d_2skw(distance from line r1 to r2)
*assumes skew lines

```
Define LibPub vector_d_2skw(r1,r2,t1,t2)=
Func
:Local normalcross,d1,d2,a1,a2
```

```
:d1:=vectore_equ(r1,t1)
:d2:=vectore_equ(r2,t2)
:normalcross:=crossP(d1,d2)
:normalcross:=unitV(normalcross)
:a1:=r1-d1
:a2:=r2-d2
:Return abs(dotP(a1-a2,normalcross))
:EndFunc
```

$vector\_d\_2skw([1+2\cdot l \quad 1-l \quad l],[2+3\cdot m \quad 1-5\cdot m \quad -1+2\cdot m],l,m)$

$a:[1 \quad 1 \quad 0]$

$d:[2\cdot l \quad -l \quad l]$

$a:[2 \quad 1 \quad -1]$

$d:[3\cdot m \quad -5\cdot m \quad 2\cdot m]$

$d1Xd2[3\cdot l\cdot m \quad -l\cdot m \quad -7\cdot l\cdot m]$

$a1-a2.normalcross$

$$\frac{10\cdot\sqrt{59}}{59}$$

Vector_d_2paral(distance between 2 parallel lines)
*assumes lines are parralel

```
Define LibPub vector_d_2paral(r1,r2,t1,t2)=Func
:Local a1,a2,qb,capq
:a1:=r1-vectore_equ(r1,t1)
:a2:=r2-vectore_equ(r2,t2)
:qb:=a1-a2
:capq:=crossP(qb,unitV(((vectore_equ(r1,t1))/(t1))))
:Return norm(capq)
:EndFunc
```

$r1(t):=[4-t \quad 2+t \quad 1+3\cdot t]$ *Done*

$r2(t):=[5-t \quad 4+t \quad -2+3\cdot t]$ *Done*

$vector\_d\_2para1(r1(t),r2(t),t,t)$

$a:[4 \quad 2 \quad 1]$
$d:[-t \quad t \quad 3\cdot t]$
$a:[5 \quad 4 \quad -2]$
$d:[-t \quad t \quad 3\cdot t]$
$a1[4 \quad 2 \quad 1]$
$a2[5 \quad 4 \quad -2]$
$a2->a1[-1 \quad -2 \quad 3]$
$|(a2-a1)X(direction \quad hat)| \quad \frac{3\cdot\sqrt{110}}{11}$

$\left|\frac{3\cdot\sqrt{110}}{11}\right|$

distance_pl2pl(distance between 2 parallel planes,)
*assumes the planes are parralel,take positive

```
Define LibPub distance_pl2pl(n1,n2,an1,an2)=
Func
:Local dist1,dist2
:dist1:=((an1)/(norm(n1)))
:dist2:=((an2)/(norm(n2)))
:Disp "((an1)/(norm(n1)))",dist1
:Disp "((an2)/(norm(n2)))",dist2
:Return dist1-dist2
:EndFunc
```

$distance\_pl2pl([3 \quad 5 \quad 1],[3 \quad 5 \quad 1],-5,6)$

$((an1)/(norm(n1))) \quad \frac{-\sqrt{35}}{7}$

$((an2)/(norm(n2))) \quad \frac{6\cdot\sqrt{35}}{35}$

$\frac{-11\cdot\sqrt{35}}{35}$

Vect_Tri_Area(p1,p2,p3)(finds area of triangle from 3 points)

```
Define LibPub vect_tri_area(p1,p2,p3)=
Func
```

```
:Local v1,v2
:v1:=p1-p2
:v2:=p3-p2
:Disp "p2->p1",v1
:Disp "p2->p3",v2
:Disp "|p2->p1Xp2->p3|/2"
:Return ((norm(crossP(v1,v2)))/(2))
:EndFunc
```

$$vect\_tri\_area([1\ \ 5\ \ -2],[0\ \ 0\ \ 0],[3\ \ 5\ \ 1])$$

$$\begin{aligned} &p2->p1[1\ \ 5\ \ -2]\\ &p2->p3[3\ \ 5\ \ 1]\\ &|p2->p1Xp2->p3|/2 \end{aligned}$$

$$\frac{\sqrt{374}}{2}$$

*not uploaded yet

Vector_Pl_LAD(eq1,eq2,k1,k2) where the plane equations are equ=k and k is a constant,the parametric equation of the line of intersection of 2 cartesian plane's

*assumes variables are x,y, and z in 3d space x=t

```
Define LibPub vector_pl_lad(equ1,equ2,k1,k2)=
Func
:Local vectequ,para
:vectequ:=zeros([equ1,equ2]-[k1,k2],{z,y,x})|x=t
:para:=dotP(vectequ,[1,0,0])
:Disp para
:Disp vectequ-vectore_equ(vectequ,para),"+",vectore_dir(vectequ,para),para
:Return expand(vectequ)
:EndFunc
```

$$vector\_pl\_lad(x+2\cdot y+z,2\cdot x+3\cdot y-2\cdot z,1,-2)$$

$$\begin{aligned} &c21\\ &a:[0\ \ 4\ \ -7]\\ &d:[c21\ \ -4\cdot c21\ \ 7\cdot c21]\\ &[0\ \ 4\ \ -7]+[1\ \ -4\ \ 7]\,c21 \end{aligned}$$

$$[c21\ \ 4-4\cdot c21\ \ 7\cdot c21-7]$$

*any thing starting with Y is included in my library,reliable bit problematic doggy

(main only has lp)

Mm.tns

1-trapezium(f,a,b,width)

2-bisection(f,lelf,right,numiterations)

3-newtonsi(f,guess,numiterations)
4-newtonsy(f,guess,vdist)
5-newtons(f,guess,tolerance)

Vectors

6-**crossP(***Vector1***,** *Vector2***)** ⇒ *vector*
*7-***dotP(***Vector1***,** *Vector2***)** ⇒ *expression*
8-unitV(vect) *returns unit vector*
9-Vector ►**Polar**
41-norm magnitude of vector
10,Y-sresolute(a,b) scaler resolute of a in direction b
11,Y-vresolute(a,b) vector resolute of a in direction b
12y-wresolute(a,b) vertical component of a
13Y-vectangle(a,b) angle between a and b,cheack if acute
14Y-vectore_equ(r,var) returns vector parralel to the equation eg f([60t,500-2t],t) -> [60t,-2t]
15Y-vector_d2l_inf(r,p,var) finds distance form line r to point p
16Y-VectPlane(p1,p2,p3) finds vector plane from 3 points
17Y-distance_pl2p(point,normal,an) finds distance form point to plane,an as in a.n=r.n
40y- vector_d_2paral(r1,r2,var1,var2) s=distance between 2 parallel lines
41y- distance_pl2pl(norml1,normal2,an1,an2) distance between 2 parallel planes
18Y-vector_d_2skw(r1,r2,t1,t2) finds distance between 2 skew lines

For distance between parallel planes find the distance to a point to plane 1 and then plane 2 and find the diffrence

Calculus

19-**arcLen(***Expr1***,***Var***,***Start***,***End***)** ⇒ *expression (arc length from start to end)*
*20-***normalLine(***Expr1***,***Var***,***Point***)** ⇒ *expression*
*21-***tangentLine(***Expr1***,***Var***,***Point***)** ⇒ *expression*

General

22-**approxRational(***Expr*[**,** *Tol*]**)** ⇒ *expression (turns decimals to approximate fraction)*
*23-***completeSquare(***ExprOrEqn***,** *Var***)** ⇒ *expression or equation (compleats the squer)*
*24-***domain(***Expr1***,** *Var***)** ⇒*expression*
*25-***euler(***Expr***,** *Var***,** *depVar***,** {*Var0, VarMax*}**,** *depVar0***,** *VarStep* [**,** *eulerStep*]**)** ⇒ *matrix*
26-Expr►**exp (turns to e)**
27-expand(*Expr1* [**,** *Var*]**)** ⇒ *expression*
*28-***factor(***Expr1*[**,** *Var*]**)** ⇒ *expression*
*29-***fMax(***Expr***,** *Var***)** | *lowBound*≤*Var*≤*upBound (max if function)*
*30-***fMin(***Expr***,** *Var***)** | *lowBound*≤*Var*≤*upBound (min of function)*
31Y-area_2graph(f1,f2,var) returns total area bounded by the two graphs,will not work with stuff like vertical assymtopes,area starts from first intersept
32Y-posinteval(f,start,end,var) returns intervals where positive kind of
*37-***polyRemainder(***Poly1***,***Poly2* [**,***Var*]**)** ⇒ *expression*

Complex Number

33-**angle(***Expr1***)** ⇒ *expression (angle of complexnumber)*
*34-***cFactor(***Expr1*[**,***Var*]**)** ⇒ *expression (factorising for complex numbers)*
*35-***cPolyRoots(***Poly***,***Var***)** ⇒ *list (roots complex)*

*36-***cSolve(***Equation***, *** Var***)** ⇒ *Boolean expression (complex solve)*

Trig

38-**tCollect(***Expr1***)** ⇒ *expression*

Returns an expression in which products and integer powers of sines and cosines are converted to a linear combination of sines and cosines of multiple angles, angle sums, and angle differences. The transformation converts trigonometric polynomials into a linear combination of their harmonics.

39-tExpand(*Expr1***)** ⇒ *expression*

Returns an expression in which sines and cosines of integer-multiple angles, angle sums, and angle differences are expanded. Because of the identity (sin(x))2+(cos(x))2=1, there are many possible equivalent results. Consequently, a result might differ from a result shown in other publications.

Code

-count

-countif(*List*,*Criteria***)** ⇒ *value (? as place holder)*

-**Fill** *Expr, listVar* ⇒ *list*

-**getKey([0|1]) ⇒ returnString**

-left(*Comparison***)** ⇒ *expression*

-∆**List(***List1***)** ⇒ *list*

-**list►mat(***List* [**,** *elementsPerRow*]**)** ⇒ *matrix*