**Plotting Equations**

with(plots);

implicitplot(eqn, x = .. , y = .. );

**Plot to find intersect**

implicitplot([eqn1,eqn2], x= .. , y= .. , color = [XXX, XXX])

color and eqns must be in [ ]

if not using color, can use {} with eqns

plot([eqn1,eqn2,...], x= .. ,y= .. ,color = [...] )

**Solve two eqns with two unknowns**

soln:= solve({eqn1,eqn2},{x,y})

if has RootOf:

solnAll:=allvalues(soln)

num:

solnAllF:=evalf(sonAll)

if can’t:

first plot, then: fsolve( {eqn1, eqn2}, {x, y}, x = range around intersection )

**Sequences, sets, and lists**

**sequence** = item separated by commas

variables := x,y,z

**set** = unordered sequence enclosed in { }

**list** = ordered sequence enclosed in [ ]

soln[2,1] same as soln[2][1]

soln:=solve( {eqns} , {unknows} ) <- eqns and unknows can both be sets {....}

**Differentiation and Integration**

secondDrivative:= diff(expr, x$2)

indefinite integral:

int(expr, x)

definite integral:

int(expr, x= .. )

**Volume of revolution**

revolve around x-axis

x=r, y=f(r)\*cos(θ), z=f(r)\*sin(θ)

revolve around y-axis

x=f(r)\*cos(θ), y=f(r)\*sin(θ), z=f(r)

then:

plot3d( [x, y, z], r = 0..N, theta = 0.. 2\*Pi ) sometimes add “scaling = constrained”

example:

plot3d( [ x, curve1\*cos(theta) , curve1\*sin(theta) ] , x = range, theta = 0 ..2\*Pi, axes

= normal )

volume:=simplify( int (Pi\*curve1^2, x = range) )

plot 2 together, add “ transparency = [0.5, 0.5] “

**ODE**

ode:=diff(y(t),t) = A\*y(t)+B

steady:=solve(rhs(ode)=0, y(t))

with no initial condition:

soln:=dsolve(ode,y(t))

with initial condition:

solnWithCond:=dsolve({ode, y(0)=y0},y(t))

for **2:** solns:=dsolve({ode1,ode2, initCond},{y1(t),y2(t)})

then:

subVals:=.....

solnSub:=subs(subVals, solnWithCond)

**More ode**

constans:=.....

odeSystem:=subs(constants,{ode, initCond})

odeNumSoln:=dsolve(odeSystem,y(t),numeric)

odeNumSoln:=proc(xxx)...end proc

odeNumSoln(0)

odeNumSoln(5)

with(plots):odeplot(odeNumSoln,[t,y(t),color=green],t=0..4\*Pi)

1st derivative

D(y)(0)=...

2nd derivative

D^(2)->(y)(0)=...

**use of exp: e^x => exp(x)**