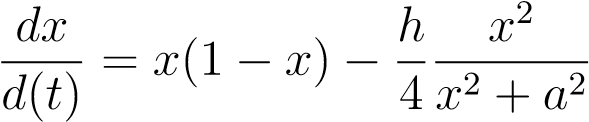
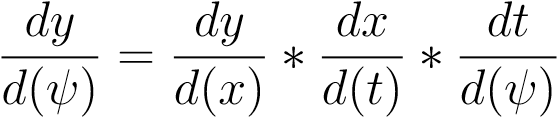
**MAT 442: HMW 1**

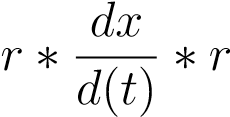
**1**. A population model is used in our scenario, which involves spruce budworm(tree-pest) and has a differential equation:

y( *ψ*) = *ry*(1 −*y/*4) −*Hy*2*/y*2 + *A*2 (1.0)

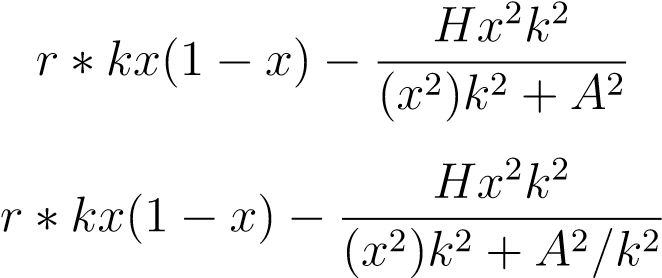
Looking at the equation above(1.0) there are four parameters namely r, K, H, A. In order to make graph plotting of our model easier, we adapt the rescaling procedure to change the dimension(number of parameters) for both population of spruce budworm and stipulated time that varies to particular threshold value.For instance, the number of spruce budworm versus needles in forest reserve which is a source of food for the spruce budworm, there is relative relationship between these two factors because the more needles consumed by spruce budworm it depletes the forest hence shortage of food supply, we can also say the same for the opposite situation.The four parameters is scaled down from four to two and parameters in equation (1.0) are assigned new variables, hence *x* = *y/k* , *a* = *A/K*, t=(*ψ*/1/r), *h* = *H/Hc* .The following steps shows how rescaled model is obtained:



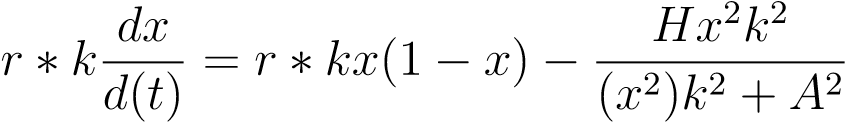
For left hand side(LHS):



For right hand side(RHS):



Therefore, LHS=RHS this implies:



Hence,

**2**. The following graphs are the reproduce Figures 2.4, 2.5 and 2.12 on pages 33 and 36 in our textboook.

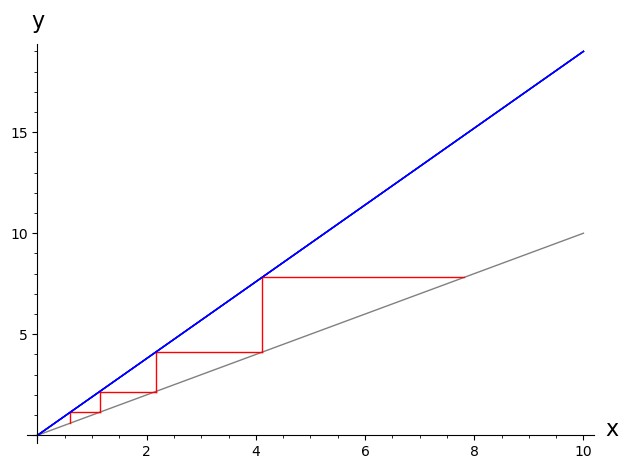


Figure 1: Cobwebbing Method, *r >* 1

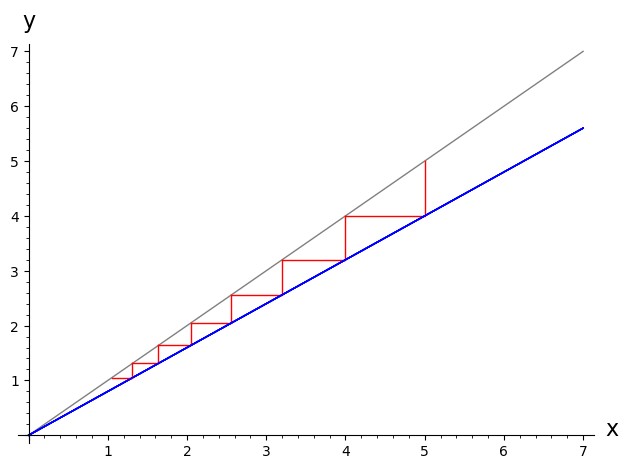
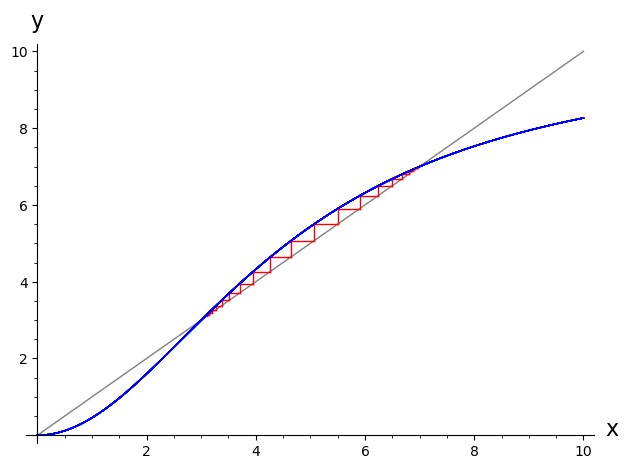


Figure 2: Cobwebbing Method, 0 *< r <* 1



√

Figure 3: Cobwebbing Method, *r >* 2 *A*

# A Appendix

## A.1 Codes

def recur(f, x0, n): points = [(0, x0)] for i in [1..n]: points.append((i, f(points[i-1][1]))) return points f(x) = 2.1\*x def snail(f,x,u0,n,xmin,xmax): u = u0

P = plot(x, x, xmin, xmax, color=’gray’) for i in range(n):

P += line([[u,u],[u,f(u)],[f(u),f(u)]], color = ’red’)

u = f(u)

P += f.plot(x, xmin, xmax, color=’blue’) P.show() snail(f,x,0.5,4,0,10)

## A.2 Codes

def recur(f, x0, n): points = [(0, x0)] for i in [1..n]: points.append((i, f(points[i-1][1]))) return points f(x) = 0.7\*x def snail(f,x,u0,n,xmin,xmax):

u = u0

P = plot(x, x, xmin, xmax, color=’gray’) for i in range(n):

P += line([[u,u],[u,f(u)],[f(u),f(u)]], color = ’red’)

u = f(u)

P += f.plot(x, xmin, xmax, color=’blue’) P.show() snail(f,x,0.9,6,0,10)

## A.3 Codes

def recur(f, x0, n): points = [(0, x0)] for i in [1..n]: points.append((i, f(points[i-1][1]))) return points f(x) = 10 ∗*x*2*/*(*x*2 + 1) def snail(f,x,u0,n,xmin,xmax):

u = u0

P = plot(x, x, xmin, xmax, color=’gray’) for i in range(n):

P += line([[u,u],[u,f(u)],[f(u),f(u)]], color = ’red’)

u = f(u)

P += f.plot(x, xmin, xmax, color=’blue’) P.show() snail(f,x,3.2,9,0,10)