Chapter –III
Monte Carlo Computations vs. Stochastic Simulation

Email:- begna19mrblgo@gmail.com

Introduction

Monte Carlo simulation

- is a technique used to understand the impact of risk and uncertainty in financial, project management, cost, and other forecasting models.
  A **Monte Carlo simulator** helps one visualize most or all of the potential outcomes to have a better idea regarding the risk of a decision.
- Monte Carlo is an estimation procedure.
- use random numbers in some way, in order to solve a model that is deterministic.
- Take for instance the classic example of Monte carlo: calculating surface of a circle (image above).
- The problem is clearly well defined and deterministic, there is only one possible outcome. Yet by sampling randomly you can approximate the answer.

Monte Carlo simulation

- A Monte Carlo approach to evaluation of these response distributions consists of the following steps:
- **Model any aspect of uncertainty** about either the input variables or the parameters of the transfer function by use of the concept of random variables. For example, the joint spatial distribution of the three variables porosity, oil saturation, and indicator of formation presence can be modeled by three, usually interdependent, random functions.
- **Draw joint realizations (outcomes)** of these random variables or functions. Each realization represents an alternative equal probable input set to the transfer function.
- **Transfer the input uncertainty** through the transfer function into sets of response values. The histogram of the response values provides a probabilistic assessment of the impact of input uncertainty on that response.

Stochastic Simulation

- is a **simulation** that traces the evolution of variables that can change stochastically (randomly) with certain probabilities.
- With a **stochastic** model we create a projection which is based on a set of random values.
- are simulations of a model that is inherently random.
- is a tool that allows Monte Carlo analysis of spatially distributed input variables.
- It aims at providing joint outcomes of any set of dependent random variables.

Stochastic simulation

These random variables can be

- Discrete (indicating the presence or absence of a character), such as facies type
- Continuous, such as porosity or permeability values
- Random sets, such as ellipses with a given distribution of size and aspect ratio, or shapes drawn at random from a frequency table of recorded shapes
  - An example would be a random number generator, giving you a number
- between 1 and 6 representing the number of eyes on a die.
  - In other words, you are simulating a random throw of the die by generating a random number.

Simulation of Pure Pursuit Problem

A fighter aircraft sights an enemy bomber and flies directly toward it, in order to catch up and destroy it.

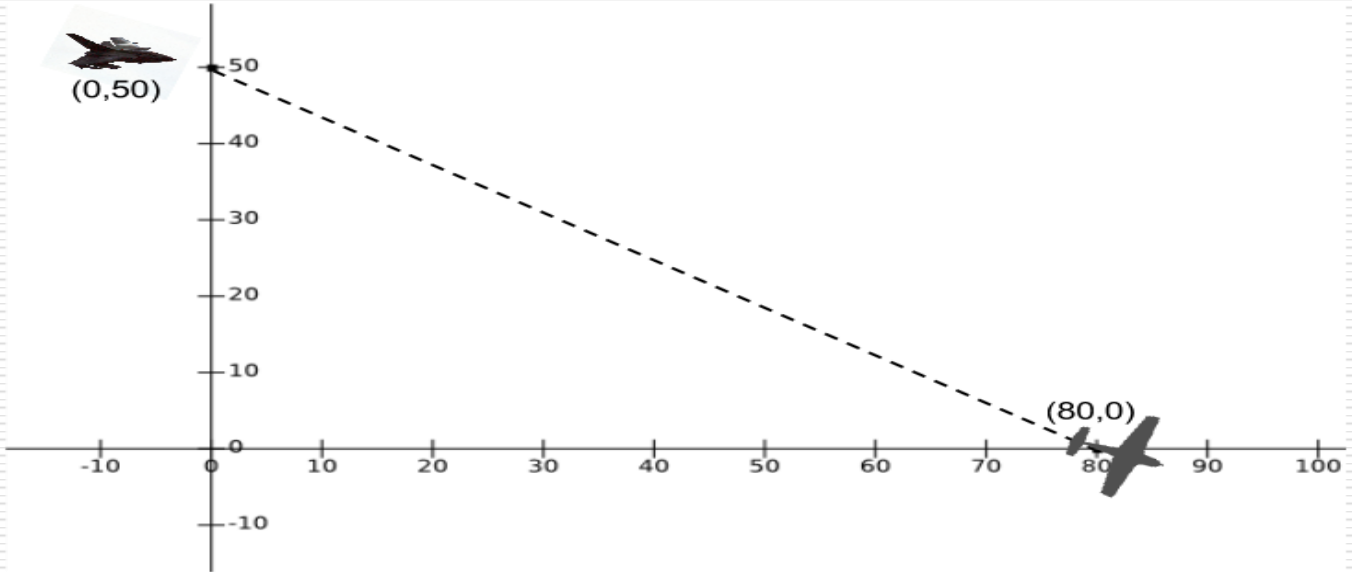
We have to determine the attack course of the fighter and how long does the fighter take to catch up with the bomber.

Cont..

- We are given following conditions:
  Both target and pursuer are flying in the same 2 dimensional plane.
- The fighter's speed is constant that is VF.
- The target's path is known.
- Minimum distance required by the fighter to fire a missile at bomber is 10 units.
- If the target is not caught within given time t_o (here t_o = 12), the target escapes.
- Initial coordinates of the pursuer (fighter) are known.

Cont..

We will choose our coordinate system such that, initially target will be lying on the X axis and pursuer will be lying on Y axis as shown in the figure below.



Positions of pursuer and target at time t=0

Simulation of a pure pursuit problem - An example

- A fighter aircraft sights an enemy bomber and flies directly towards it in order to catch up bomber and destroy it.
- Pure Pursuit - The simple strategy of pursuer redirecting himself toward the target at fixed intervals of time, while the target goes on its predetermined path without making any effort to evade the pursuer.
- If path of the target is straight then problem can be solved directly using analytical techniques.
- However, if path of the target is curved then problem becomes more complex and cannot be solved directly but through simulation only.
- Conditions to simulate must be well specified first.

Simulation of a pure pursuit problem - An example

- **Some formulae used here:**
- Dist(t) = sqrt((yb(t)-yf(t))exp2+(xb(t)-xf(t))exp2)

  Sin (theta) = yb(t)-yf(t) / dist(t)
  Cos(theta) = xb(t) - yf(t) / dist (t)

Cont..

- Given position at time 't', next position at time 't+1' is computed using:

  $Xf(t+1) = xf(t) + vf*\cos(theta)$
  $Yf(t+1) = yf(t) + vf*\sin(theta)$

  Analytically we cannot make a long term prediction about the path that the fighter plane would take (given the initial position and path of the target).

- But by Simulation, we were able to make instant-to-instant predictions for as many instants as we wanted.

Cont..

We are given following conditions:
Both target and pursuer are flying in the same 2 dimensional plane.

- The fighter's speed is constant that is VF.
- The target's path is known.
- Minimum distance required by the fighter to fire a missile at bomber is 10 units.
- If the target is not caught within given time t_o (here t_o = 12), the target escapes.
- Initial coordinates of the pursuer (fighter) are known.

Pursuit Algorithm

The implementation of the pure pursuit algorithm itself is fairly straightforward.

The pure pursuit algorithm can be outlined as follows: -

- Determine the current location of the vehicle.
- Find the path point closest to the vehicle.
- Find the goal point
- Transform the goal point to vehicle coordinates.
- Calculate the curvature and request the vehicle to set the steering to that curvature.
- Update the vehicle’s position.

Cont..

```
package simulation;
 public class Simulation {
public static void main(String[] args) {
 /*Co-ordinates of the bomber*/
double[] xb={80,90,99,108,116,125,133,141,151,160,169,179,180};
 double[] yb={0,-2,-5,-9,-15,-18,-23,-29,-28,-25,-21,-20,-17};
 /*Co_ordinates of fighter to be determined*/
double[] xf=new double[13];
double[] yf=new double[13];
xf[0]=0;
 yf[0]=50;
double vf=20; //speed of fighter
boolean status=true;
double dist;
for (int t=0;t<12;t++)
{
dist=Math.sqrt((xb[t]-xf[t])*(xb[t]-xf[t])+(yb[t]-yf[t])*(yb[t]-yf[t]));
if(dist<=10) { status=false;
for(int i=0;i<13;i++) {
System.out.println(xf[i]+" "+yf[i]);
}
```

Cont..

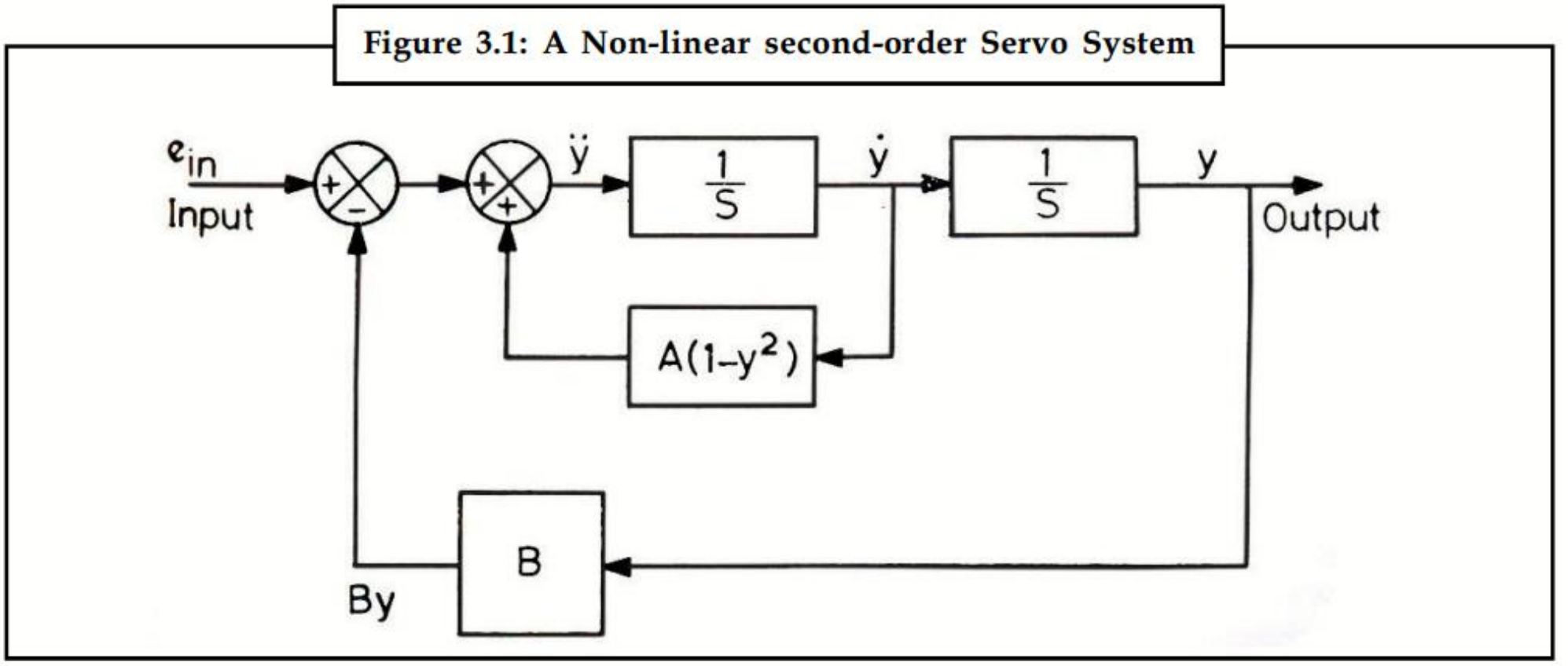
```
System.out.println("Target Destroyed at time t="+t);
break;
}
yf[t+1]=yf[t]+vf*((yb[t]-yf[t])/dist);
xf[t+1]=xf[t]+vf*((xb[t]-xf[t])/dist); }
 if(status) {
for(int i=0;i<13;i++) {
 System.out.println(xf[i]+" "+yf[i]);
 } System.out.println("Target escaped!");
}
}
}
```

Simulation of a Servo System

A very important application of continuous system simulation is in design and analysis of control systems.

Let us study the behavior of a second order nonlinear feedback system represented by the following block diagram.

Simulation of a Servo System



SIMULATION OF A SERVO SYSTEM USING VAN DER POL NON-LINEAR EQUATION

A servo system can be described by the following second order differential equation:

$$(d^2y/dt^2) = P\,(1-y^2)*(dy/dt) - Qy$$

Where P and Q are positive constants, y is the position coordinate which is a function of the time t.

This differential equation is the well-known Van der Pol non-linear equation. To simulate this system , this second order differential equation is to be written as a set of two simultaneous equations of first order as follows:

By using the variable y1 in place of y, we get

$$dy1/dt = y2$$
$$dy2/dt = P(1- y1^2)* y2 - Qy1$$

Cont..

Assuming constants P to be 0.1 and Q =1.0 and the initial conditions to be y1(0)=1.0 and y2(0)=0, our equation becomes:

$$dy1/dt = y2$$
$$dy2/dt = 0.1(1- y1^2)*y2- y1$$

The system has been simulated for 20 seconds with a step size of 0.001 seconds.
That is, 20000 computations have been done. The output is printed once for every 100 integration steps and the same has been plotted on graph.

Source code

```
#include < stdio.h >
#include < conio.h >
#include < math.h >
#include < stdlib.h >

void main()
 {
    clrscr();
    float t=0,y1=1,y2=0,h=0.001,u11,u12,u21,u22,u31,u32,u41,u42;
    long i;
    for(i=1;i<=20000;i++)
      {
//Runge kutta's term
       u11=h*y2;
       u12=h*(0.1*(1-y1*y1)*y2-y1);
       u21=h*(y2+0.5*u12);
       u22=h*(0.1*(1-(y1+0.5*u11)*(y1+0.5*u11))*(y2+0.5*u12)-(y1+0.5*u11));
```

cont

```
u31=h*(y2+0.5*u22);
 u32=h*(0.1*(1-(y1+0.5*u21)*(y1+0.5*u21))*(y2+0.5*u22)-
(y1+0.5*u21));
      u41=h*(y2+u32);
 u42=h*(0.1*(1-(y1+u31)*(y1+u31))*(y2+u32)-(y1+u31));
      y1=y1+(u11+2*u21+2*u31+u41)/6;
      y2=y2+(u12+2*u22+2*u32+u42)/6;
      t=t+h;
 if(i%100==0)
      printf("%f    %f    %f \n", t,y1,y2)
   }
  getch();
}
```

Simulation of Chemical Reactor

In a certain chemical reaction when two substances A and B are brought together.

They produce a substance C. It is known that 1 gm of A and 1 gm of B produce 2gms of C.

Rate of formation of C is proportional to amounts of A and B present.

In addition to this forward reaction there is also a backward reaction decomposing C back to A and B.

The rate of decomposition is proportional to amount of C present in the mixture.

Cont..

In other words at any time if a, b, c are the amounts of A, B, C the following differential equations express the rates of increases:

$da/dt = k2c - k1ab$

$db/dt = k2c - k1ab$

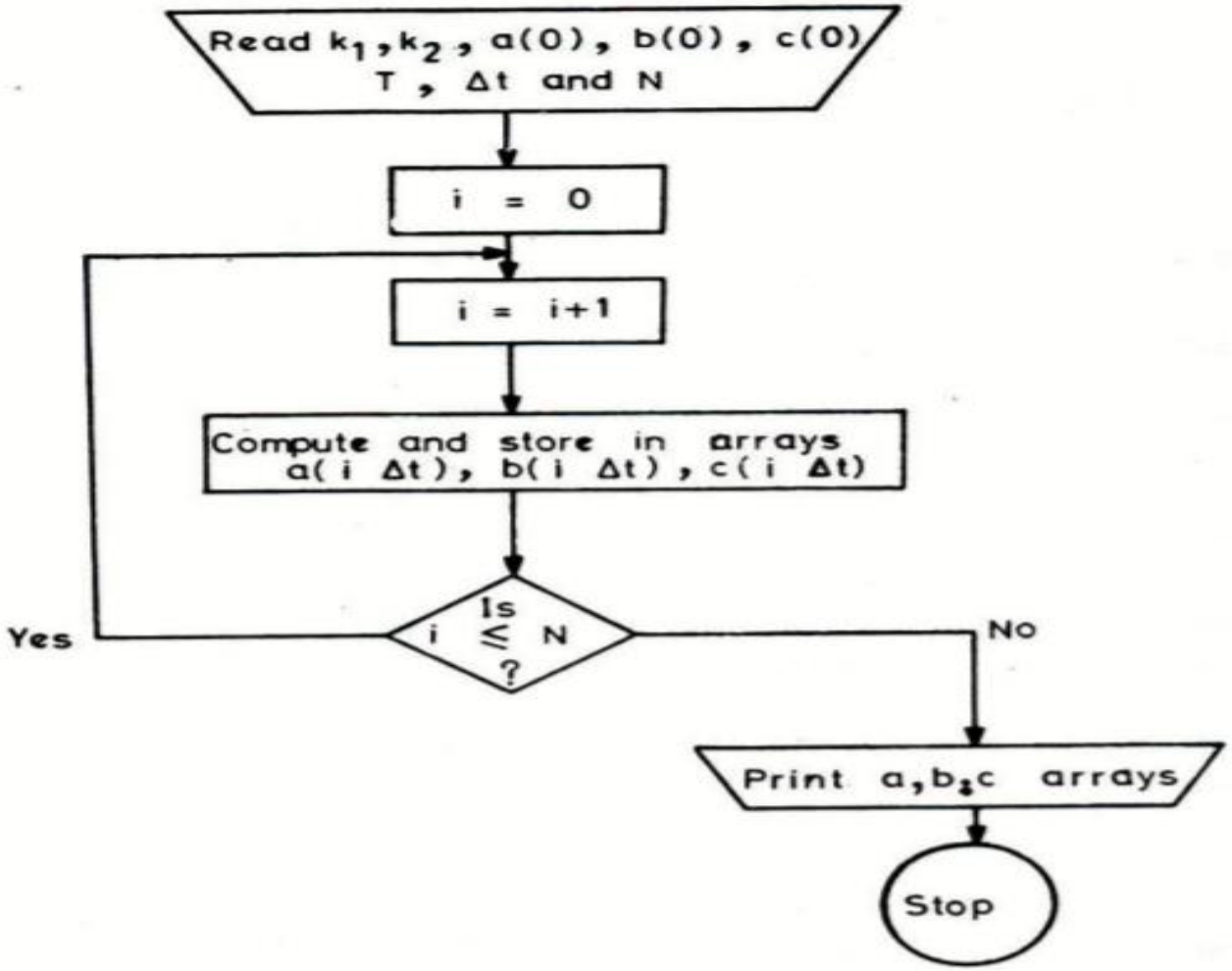
$dc/dt = 2k1k2ab - 2k2c$ where k1, k2 are rate constants

The constants k1, k2 vary with temperature and pressure.

Given k1, k2 and initial quantities of A and B we wish to determine how much of C has been produced at any time.

Such determinations of rates of chemical reactions are important in many industrial applications.

Simulation of Chemical Reactor



Simulation of Single Server

Note: for academic purpose, all the computations (i.e. computation of Cumulative Arrival Time,Cumulative Departure Time, Queue Length, Idle Time and Waiting Time) have been carried out using different loops.

Abbreviations used:at: Arrival Time, st: Service Time, cat: Cumulative Arrival Time, cdt: Cumulative Departure Time, ql: Queue Length, idt: Idle Time, wt: Waiting Time.

Assumptions: Time gap between the arrival of any two consecutive customers (at) is between 5 to 30 minutes

Service Time is from 5 to 35 minutes

Simulation has been done for 20 customers

Cont.

```
#include < stdio.h >
#include < conio.h >
#include < math.h >
#include < stdlib.h >
void main()
 {
    clrscr();
    int at[20],st[20],cat[20],cdt[20],ql[20],idt[20],wt[20];
    int i,j,k;
    cat[0]=0,wt[0]=0,idt[0]=0;
    randomize();
    for(k=0;k<=19;k++)
      {
        at[k]=(5+rand()%25);
        st[k]=(5+rand()%30);
        ql[k]=0;
      }
     at[0]=0;
     cdt[0]=st[0];
     for(j=1;j<=19;j++)
     {
        cat[j]=cat[j-1]+at[j];
        if(cat[j]<=cdt[j-1])
           cdt[j]=cdt[j-1]+st[j];
       else
```

Cont..

```
cdt[j]=cat[j]+st[j];
    k=j-1;
    while(cdt[k] > cat[j] && k>=0)
     {
       ql[j]=ql[j]+1;
        k=k-1;
     }
     if(cat[j]<=cdt[j-1])
     {
       wt[j]=cdt[j-1]-cat[j];
       idt[j]=0;
     }
    else
     {
        idt[j]=cat[j]-cdt[j-1];
        wt[j]=0;
     }
     }
```

Cont..

```
printf(" AT    ST    CAT    CDT     QL    IDT    WT \n\n\n");
    for(k=0;k<=19;k++)
      {
       printf("%3d   %3d    %3d    %3d    %3d    %3d   %3d",at[k],st[k],cat[k],c
dt[k],ql[k],idt[k],wt[k]);
        printf("\n\n");
       }
    getch();
     }
```

Multi Server Queue

- As the name suggests, the system consists of multiple servers and a common queue for all items.
- When any item requests for the server, it is allocated if at-least one server is available.
- Else the queue begins to start until the server is free.
- In this system, we assume that all servers are identical, i.e. there is no difference which server is chosen for which item.

Multi Server Queue

- There is an exception of utilization. Let **N** be the identical servers, then **ρ** is the utilization of each server.
- Consider **Nρ** to be the utilization of the entire system; then the maximum utilization is **N*100%**, and the maximum input rate is −

$$\lambda max = NTs \lambda max = NTs$$

Simulation Of Inventory Model

- The Inventory management is one of the crucial aspects for any manufacturing firm and well known topic in both corporate and academic world.
- Inventory management involves a set of decisions that aim at matching **existing demand** with the supply of products and materials over **space** and **time**.
- Inventory management is important because it can give answer to firms about when to order, how much to order and how much stock to keep as safety stock.
- Another objective of inventory management is to minimize cost while maintaining acceptable service level.

Simulation Of Inventory Model

- Inventory management model selection focused on production and distribution environments in which demand and lead time tend to be more predictable.
- The software makes the use of the simulation relatively unstructured and interactive, so user can control the time and frequency of usage.
- User can form a computer programming model for the existing inventory system and run simulation on it to predict the provable situation of the system.

Simulation Of Inventory Model

```
#include < stdio.h >
#include < conio.h >
#include < math.h >
#include < stdlib.h >
void main()
 {
    clrscr();
    int day,rday=0,flag=0, stock=80,demand,reorder;
    int  profit=0, cost=0;
    randomize();// to initialize random numbers
    printf("Initial stock is %d\n\n", stock);
    printf("Day    Demand     Profit    Cost    Bal.Stock\n\n");
    for(day=1; day<=30; day++)
    {
      if((day==(rday+3)) && flag==1)
     {
      stock= stock+reorder;
      printf("%d units have been delivered and has been added to the stock\n\n",reorder);
      printf("Current stock (on day %d) is %d\n\n", day,stock);
      printf("Day    Demand     Profit    Cost    Bal. Stock\n\n");
      flag=0;
     }
      demand=rand()%30;
      if (demand<=stock)
    {
```

Cont..

```
        profit = profit+ demand*20;
        cost=cost+(stock-demand)*2;
        stock=stock-demand;
        }
          else
        {
        profit=profit+stock*20;
        cost= cost+(demand-stock)*22;
        stock=0;
        }
          printf("%d      %d        %d       %d       %d\n\n", day,demand,profit,cost,stock);
          /*getch();*/
          if(stock<=50 && flag==0)
         {
        reorder= 50+rand()%30;
        cost=cost+75;
        rday=day;
        flag=1;
        printf("Order given for %d units at the end of day %d \n\n",reorder,day);
        printf("This will be delivered on day %d \n\n",day+3);
        printf("Day    Demand     Profit    Cost    Bal.Stock\n\n");
          }

      }
      printf("\n\nTotal Profit is %d\n\n", profit);
      printf("Total Cost (Loss of Good Will+ Carrying Cost + Reorder Cost) is %d\n\n", cost);
      getch();
    }
```

Simulation Of A Water Reservoir System

- While simulating this system, the following assumptions are made:
  1. Simulation is done for 60 consecutive months
  2. Water received through direct Rain Fall is assumed to be not more than 30% of the capacity of the reservoir
  3. Maximum water received through River Inflow is 130% of the capacity of the reservoir

Simulation Of A Water Reservoir System

4. Seepage loss is not more than 5% and Evaporation loss is not more than 8% of the gross volume of water

5. Demand is never more than the Capacity of the Reservoir.

Simulation Of A Water Reservoir System

```
#include < stdio.h >
#include < conio.h >
#include < math.h >
#include < stdlib.h >
void main()
 {
    clrscr();
    float rain[61],rflow[61],vin[61],seep[61],evap[61],tloss[61],vnet[61];
    float grossv[61],shortage[61],spill[61],vol[61],dem[61];
    float dif;
    int m,cap=100;
    vol[0]=0;
    randomize();
```

Simulation Of A Water Reservoir System

```
for(m=1;m<=60;m++)
      {
 rain[m] =(rand()%cap)*0.3;
 rflow[m]=(rand()%cap)*1.3;
 vin[m]=rain[m]+rflow[m];
 grossv[m]=vol[m-1]+vin[m];
 seep[m]= grossv[m]*0.05;
 evap[m]=grossv[m]*0.08;
 tloss[m]=seep[m]+evap[m];
 dem[m]=rand()%cap;
if(tloss[m]>grossv[m])
   {
     shortage[m]=dem[m];
     vol[m]=0;
     spill[m]=0;
   }
```

Simulation Of A Water Reservoir System

```
      else
{
 vnet[m]=grossv[m]-tloss[m];
 if (dem[m]>vnet[m])
    shortage[m]=dem[m]-vnet[m];
 else
 {
   dif=vnet[m]-dem[m];
   if (dif>cap)
    {
     spill[m]=dif-cap;
     vol[m]=cap;
     shortage[m]=0;
    }
 else
   {
   vol[m]=dif;
   spill[m]=0;
   shortage[m]=0;
   }
}
```

Simulation Of A Water Reservoir System

```
printf("Mon Rain  Rflow   Vin   V[m-1] Grossv   Seep  Evap  Tloss Vnet  Dem  Short
Spill\n\n");
      for(m=1;m<=60;m++)
      {
 printf("%2.0d %5.1f %6.1f %6.1f %6.1f %6.1f %6.1f %6.1f %5.1f %5.1f %5.1f
%5.1f %5.1f %5.1f\n\n",m,rain[m],rflow[m],vin[m],vol[m-
1],grossv[m],seep[m],evap[m],tloss[m],vnet[m],dem[m],shortage[m],spill[m]);
 if((m%15)==0 &&(m<60 p=""> {
 getch();
 printf("Mon Rain  Rflow   Vin   V[m-1] Grossv   Seep  Evap  Tloss Vnet  Dem  Short
Spill\n\n");
 }
      }
      getch();
    }
```