Problem 1

```
restart :
```

I am loading packages I need for this problem. Plots for display and pointplot options. LinearAlgebra for Array option because I want to work with Arrays/Matrix (they are so nice to work with becasue it stores all the info I need).

```
with(plots) :
with(LinearAlgebra) :
with(ColorTools) :
```

I initilize my formulas. r is the distance between particles 1 and 2, where we can do any 2 particles. And Gx and Gy are the force on the particle in y and x axis. We want to work separately on each coordiate. I checked that it workes with numbers. In the handwritten write up I show this with some drowings and little derivation.

$$r := (x1, x2, y1, y2) \rightarrow \mathrm{sqrt}\left((x1 - x2)^2 + (y1 - y2)^2 \right) :$$

$$Gx := (x1, x2, y1, y2) \rightarrow -\frac{1}{r(x1, x2, y1, y2)^2} \cdot (x1 - x2) :$$

$$Gy := (x1, x2, y1, y2) \rightarrow -\frac{1}{r(x1, x2, y1, y2)^2} \cdot (y1 - y2) :$$

I create a variable that makes a random number. What range it is does not really matter becasue I just need a random number to fill me matrix with them. The reason I am not using just some constant numebr is that becasue then my distance happen to be 0 if I give all the particles the same position. Does not matter for the V (velocity) matrix so I just used same number to save the speed and memory of the computer. (Which in this case is like no difference but still it is nice to start thinking about that).

N is the number of particles that we are using.

IR - initial position of each particle and IV - initial velosity of each particle.

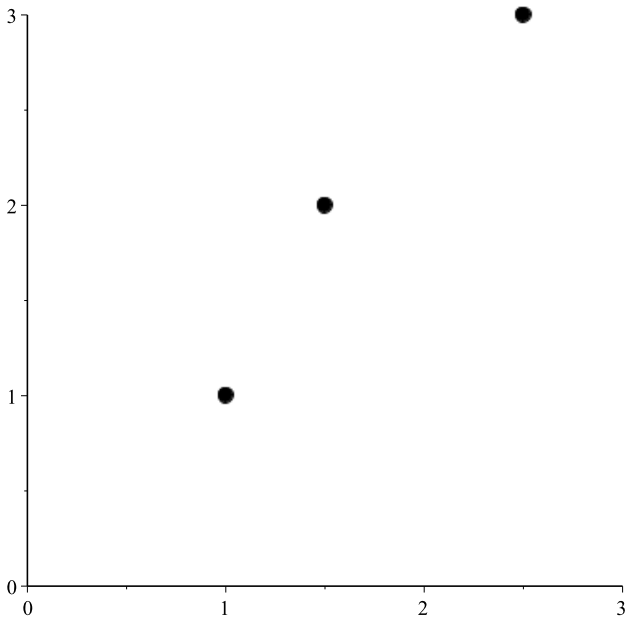
I use this notion of a list within a list. Where each list item (which is information for each particle) is a list with 2 items (x and y component for that particulat particle).

I also plot the initial position for the particles.

```
rr := rand(0..100) :
IR := [[[1, 1], [1.5, 2], [2.5, 3]]] :
IV := [[[0, 0.8], [1, 1.4], [0.2, -1]]] :
N := nops(IR[1])
```

3 **(1)**

```
pointplot(IR[1], view = [0..3, 0..3], symbol = solidcircle, symbolsize = 20)
```



I Initilize the condition for the loop. We have a step thau of 0.1 second and we will compute the position for 10 seconds. Each step is a new postition for the particle. Therefore for each particle we will have 100 positions

$thau := 0.1 :$

$T := 10 :$

$Nt := \frac{T}{thau}$

100.0000000 **(2)**

I create 2 Arrays for Position and Velocity of particles. Arrays have 100 rows (where each row coreesponds for each step at a time. Each row stores information for every 0.1 second (in our case) for each particle position (and velocity in the other Array), I keep the row number as my variable Nt which depends on time and step that we choose. Each Array has 3 colums, where amount of colums correspind to numebr of particles that we have (N variable). Each [row][column] psition stores information about particular particle position/velociy at some time. However each partivle has x and y component. So each [row][colums] position stores another list with 2 items x and y component. In the handwritten write up I show how it lookes like. Maple does not want to show that.

```
XX := Array([IR[1], op([seq([seq([rr(), rr()], k = 1..N)], i = 1..Nt)])):
VV := Array([IV[1], op([seq([seq([1, 2], k = 1..N)], i = 1..Nt)])):
```

This is my loop and I color it with different color and comment with correspondind color to explain what is happening there.

I make sure the variable I loop over are not initilized or anything.

Since there are multiple particles acting on the particular particle the total force is the sum of the forces. This loop does the summation over all the fores acting on the prticulat particle. i(for position) and j(for velosity) is the numebr of that particular particle and k is the number of all the particles around that particulat particle. So we sum all the forces exept the force on itself (if statement takes care of that). In the handwritten write up I graphicly show why sum of forces is important.

For the velosity we do a different force summation but in the very similar algoritm. The only diffrenece is that accordinfg to Verlet Algoritm we need froce at time t and t+1.

This loop is taking care so we do these summations for each particle and calculate the position (i loop)/ velosity (j loop) for each particle at particular time. We do sumation for one particle calculate position/ velosity. And do this for all particles at that row (time).

Repeat everything stated above Nt times (however many time steps we have). Time goes from 1 to Nt-1 becasue the last thing we do is calculate and place the postion/velocity of particle at time t+1.

Note. I have to do part with position and velosity separetaly becasue we use position to calculate velosity. So we have to find position for all particles and then we find velosity at that time for all particles.

```
t := 't': i := 'i': j := 'j': k := 'k':
for t from 1 to Nt − 1 do
  for i from 1 to N do
    SumX := 0;
    SumY := 0;
    for k from 1 to N do
      if i ≠ k then
        SumX := SumX + evalf(Gx(XX[t, i, 1], XX[t, k, 1], XX[t, i, 2], XX[t, k, 2]));
        SumY := SumY + evalf(Gy(XX[t, i, 1], XX[t, k, 1], XX[t, i, 2], XX[t, k, 2]));
      end if
    end do:
        XX[t + 1, i, 1] := XX[t, i, 1] + thau·VV[t, i, 1] + thau^2/2 ·SumX;
        XX[t + 1, i, 2] := XX[t, i, 2] + thau·VV[t, i, 2] + thau^2/2 ·SumY;
  end do:
  for j from 1 to N do
    SumVY := 0;
    SumVX := 0;
    for k from 1 to N do
      if j ≠ k then
        SumVX := SumVX + evalf(Gx(XX[t, j, 1], XX[t, k, 1], XX[t, j, 2], XX[t, k, 2]) + Gx(XX[t
    + 1, j, 1], XX[t + 1, k, 1], XX[t + 1, j, 2], XX[t + 1, k, 2]));
        SumVY := SumVY + evalf(Gy(XX[t, j, 1], XX[t, k, 1], XX[t, j, 2], XX[t, k, 2]) + Gy(XX[t
```

$+1, j, 1], XX[t+1, k, 1], XX[t+1, j, 2], XX[t+1, k, 2]));$

end if

end do:

$$VV[t+1, j, 1] := VV[t, j, 1] + \frac{thau}{2} \cdot SumVX;$$

$$VV[t+1, j, 2] := VV[t, j, 2] + \frac{thau}{2} \cdot SumVY;$$

end do

end do:

Now I want to plot and see how my particle moves. Each step is a new position so plotting each step as a point we get the path of each particle.

I want each path for each particle be its own color. So I make a loop that cretes a 1xN matrix or random colors. I leave it so it shows what colors it made so in case it randomly created similar color I rerun the loop. Then when I plot, for each number particle there is a corresponding nuber color that is indexed as an item of col list.

$col := RandomMatrix(1, N):$

$$rrr := evalf\left(\frac{rand(0..100)}{100}\right):$$

for c **from** 1 **to** N **do**

$col[1, c] := Color([rrr(), rrr(), rrr()]);$

end do

⟨RGB 0.87 0.11 0.55⟩

⟨RGB 0.35 0.92 0.83⟩

⟨RGB 0.51 0.39 0.54⟩

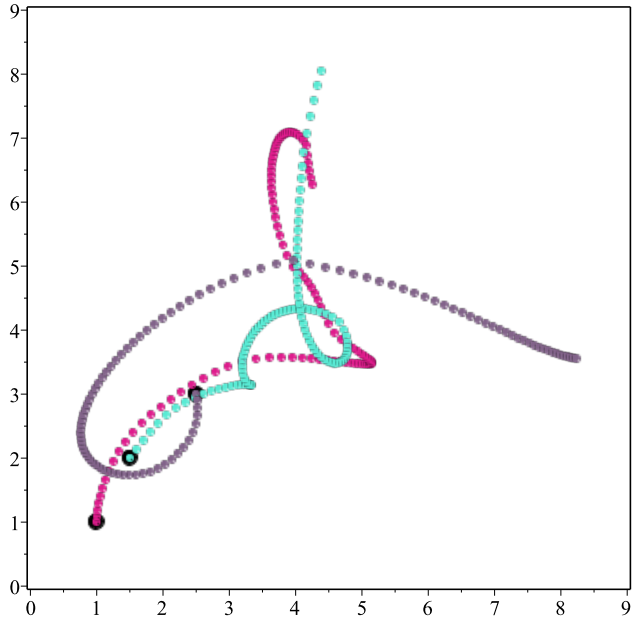
$IA := display(seq(pointplot(XX[1, j], color = black, symbolsize = 20, symbol = solidcircle), j = 1..N))$

$PLOT(...)$

$IB := display(seq(seq(pointplot(XX[i, j], color = col[1, j], symbolsize = 10, symbol = solidcircle), i = 1..Nt), j = 1..N))$

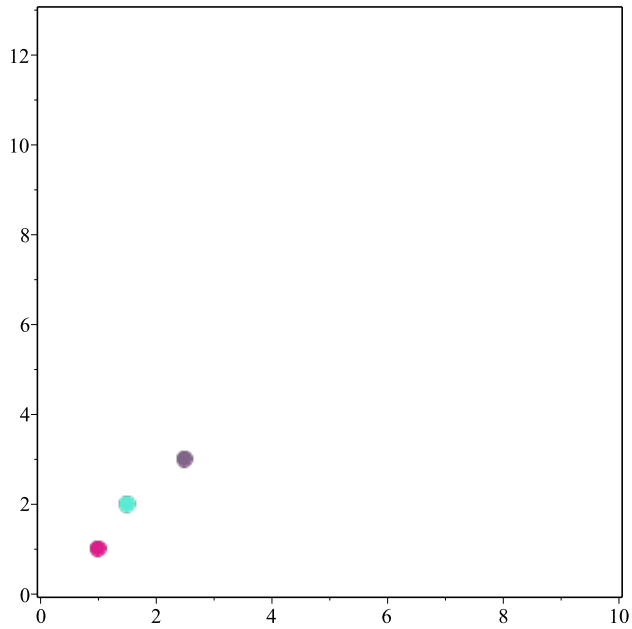
$PLOT(...)$ **(5)**

$display(IA, IB, axes = boxed, view = [0..9, 0..9])$



I also made and autameted an animation of the three particles. Keeping them same color as the plot above. And it automated so it will run for whatever many paeticles we need without changing anything just initial condition of however many particles we want.

display(*seq*(*display*(*seq*(*pointplot*(XX[*i*, *j*]), i = 1 ..Nt), *color* = *col*[1, *j*], *axes* = *boxed*, *view* = [0 .. 10 , 0 ..13], *symbol* = *solidcircle*, *symbolsize* = 20, *insequence* = *true*), *j* = 1 ..*N*))



That was a very fun problem. Enjoyed "playing" with it and automate everything and make it easy to see on plots at the end. I feel like this algoritm would work for whatever many particles where in the beggining just change initial conditions for whatever many particles we want.

Problem 2

a

I make sure t (time variable) is not initilized.

$t := 't'$:

Now I set up my 6 diffirential equations. Base for this differential equation is F=ma - this is what we solve, where m=1. a - acceleration is the double derivative of position with respect to time. F is the total force on the particle (I use same force formulas, and just add them manualy. We have only 3 particles so that is not bad). We put them all together because they all depend on each other.

$$ode := \frac{d^2}{dt^2} xa(t) = (Gx(xa(t), xb(t), ya(t), yb(t)) + Gx(xa(t), xc(t), ya(t), yc(t))), \frac{d^2}{dt^2} xb(t)$$

$$= (Gx(xb(t), xa(t), yb(t), ya(t)) + Gx(xb(t), xc(t), yb(t), yc(t))), \frac{d^2}{dt^2} xc(t) = (Gx(xc(t),$$

$$xa(t), yc(t), ya(t)) + Gx(xc(t), xb(t), yc(t), yb(t))), \frac{d^2}{dt^2} ya(t) = (Gy(xa(t), xb(t), ya(t), yb(t))$$

$$+ Gy(xa(t), xc(t), ya(t), yc(t))), \frac{d^2}{dt^2} yb(t) = (Gy(xb(t), xa(t), yb(t), ya(t)) + Gy(xb(t),$$

$$xc(t), yb(t), yc(t))), \frac{d^2}{dt^2} yc(t) = (Gy(xc(t), xa(t), yc(t), ya(t)) + Gy(xc(t), xb(t), yc(t),$$

$$yb(t))) :$$

I make a variable with initial conditions, where I take numbers form already initilized list of initial condition IR and IV.

$$inc := xa(0) = IR[1, 1, 1], xb(0) = IR[1, 2, 1], xc(0) = IR[1, 3, 1], ya(0) = IR[1, 1, 2], yb(0) = IR[1, 2, 2], yc(0) = IR[1, 3, 2], xa'(0) = IV[1, 1, 1], xb'(0) = IV[1, 2, 1], xc'(0) = IV[1, 3, 1], ya'(0) = IV[1, 1, 2], yb'(0) = IV[1, 2, 2], yc'(0) = IV[1, 3, 2]$$

$$xa(0) = 1, xb(0) = 1.5, xc(0) = 2.5, ya(0) = 1, yb(0) = 2, yc(0) = 3, D(xa)(0) = 0, D(xb)(0) = 1, D(xc)(0) = 0.2, D(ya)(0) = 0.8, D(yb)(0) = 1.4, D(yc)(0) = -1 \quad \textbf{(6)}$$

Here I just solve the differential equations in numeric form and initilize each position fucntion for x and y for each particle taking pspecific part of the solution (for each particle x ,y).

```
sol1 := dsolve([ode, inc], numeric, output = listprocedure) :
fxa := eval(xa(t), sol1);
fya := eval(ya(t), sol1);
fxb := eval(xb(t), sol1);
fyb := eval(yb(t), sol1);
fxc := eval(xc(t), sol1);
fyc := eval(yc(t), sol1);
```

proc(t) ... **end proc**

proc(t) ... **end proc**

proc(t) ... **end proc**

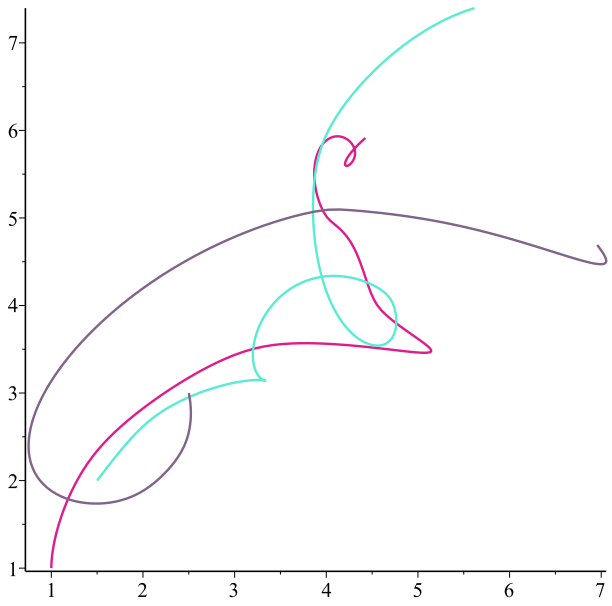
proc(t) ... **end proc**

proc(t) ... **end proc**

proc(t) ... **end proc** **(7)**

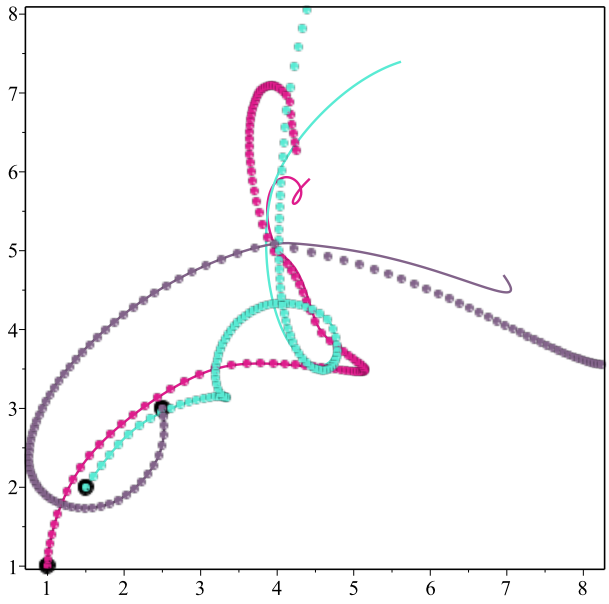
I simply plot the functions. I use same color as before in prolem 1.

```
display(plot([fxa(t), fya(t), t = 0..10], color = col[1, 1]), plot([fxb(t), fyb(t), t = 0..10], color = col[1, 2]), plot([fxc(t), fyc(t), t = 0..10], color = col[1, 3]))
```



b)I plot my solution for problem 2 and 1 together with same colors so we can track each particle.

$$display(plot([fxa(t), fya(t), t = 0..10], color = col[1, 1]), plot([fxb(t), fyb(t), t = 0..10], color = col[1, 2]), plot([fxc(t), fyc(t), t = 0..10], color = col[1, 3]), IA, IB, axes = boxed)$$



So one big difference is that problem 1 solution is descrete and approximaete and problem 2 solution is continious and exact. Two solutions agree very well in the beggining up to around half of th time and then they start to deviate from each other (bigger time more deviation). If we used smaller step they would start deviate later in time becasue smaller step is more like continious solution.
When we do a step we lose some information about the position and velocity between t1 and ti+1, therefore with each consequtive step we get a bigger error on the position and velosity and that error increses I believe exponentialy (or something like that).
Differential equation also kind of uses a step but infinetely small step. Wich such a small step so that both solutions compare it would take a long time for the loop to run.

Problem 3

So I start with restarting everything (so I don't have initilized variables I don't need) and loading packages.

```
restart :
with(plots) :
with(stats) :
with(Statistics) :
```

```
printlevel := 0 :
```

Cube has 6 sides. Probability of succes (getting 6) is 1/6. I initilize the variables ps - probability of success, n - how many times we throw the dice, k = list with the outcomes we want (1,3 or 5 times we get 6 on dice out of 15 throws).

```
ps := 1/6 :
n := 15 :
k := [1, 3, 5] :
```

I made a loop to calculate the probability of getting 1, 3 or 5 times we get 6 on dice out of 15 throws. I use binomial build on maple formula to to n choose k part and then I use it for my binomial distribution formula. Since we want to know the probability of getting 1, 3 or 5 - we want to sum the probabilites of each - where my loop does that as well. We found that calculated probability is 0.4933438928.

```
p := [0, 0, 0] :
summ := 0 :
 for i from 1 to 3 do
   p[i] := evalf(binomial(n, k[i]) · ps^k[i] · (1 − ps)^(n − k[i]));
   summ := summ + p[i] :
 end do:
summ
```

$$0.4933438928 \tag{8}$$

Now I proceed to loop calculations. I have the variable that makes a random number in between 0 and 1.

```
rr := x → stats[random, uniform[0, 1]](1)
```

$$x \rightarrow stats[random, uniform_{0,\,1}](1) \tag{9}$$

```
rr( )
```

$$0.3957188605 \tag{10}$$

There is the loop.

This part of the loop is "throwing a dice" 15 times. By throwing we basically just have a random numebr between 0 and 1. Then we see if it is smaller or equal to 1/6 (success probability). We need numebrs that are only 1/6 part of the whole (could have been greater then 5/6). Nuber is smaller then 1/6 we count it in our N counter.

This part of the loop runs the red loo imax (5000) times. After running red loop we look the numebr of N counter. If it is 1,3, or 5 we add one to H counter.

So at the end we have some number for H which shows how many times out of 5000 we got 1,3 or 5 times we get 6 on dice out of 15 throws. We devide that by imax and get our pobability.

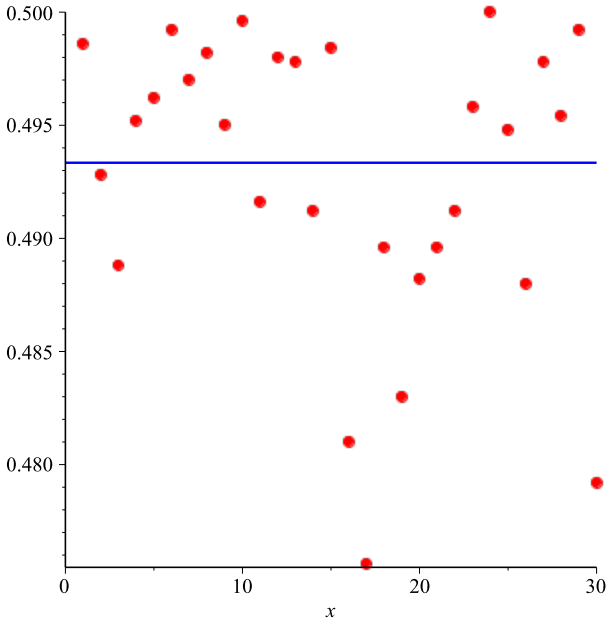
This loop runs the whole aloritm p times and creates the list of values for the probability that we can plot and compare to the binomial result.

```
imax := 5000 :
 p := 30 :
 l := [ ] :
 for k from 1 to p do
   H := 0;
```

```
for i from 1 to imax do
  N := 0 :
  for j from 1 to 15  do
    xp := evalf(rr( ));
    if xp ≤ evalf( (ps) ) then
    N := N + 1;
    end if
  end do;
  if N = 1 or N = 3 or N = 5 then
  H := H + 1;
  end if
end do;
l := [op(l), evalf(H/imax)];
end do:
```

I make a sequence where so I can plot my l items versus the # of the point. And I plot it together with the binomial distribution solution (straight line).

```
ss := [seq( [i, l[i]], i = 1 ..p) ] :
display(plot(summ, x = 0 ..p, color = blue), (pointplot(ss), color = red), symbolsize = 15, symbol
    = solidcircle)
```



We can see how some calculations we got with a loop are very close and some are actually pretty far. But in general pretty close to the binomial solution. I decided to make a loop to calculate the average percentage difference between the loop solutions and the binomial solution. I also found max percentage differnece.

$sumd := 0$:

for f **from** 1 **to** p **do**

$$sumd := sumd + \mathrm{abs}\left(\frac{(l[p] - summ)}{summ} \right) :$$

end do:

$$avg := \frac{sumd}{p} \cdot 100;$$

2.866943930 **(11)**

So for this particular run I found avarage percentage difference is 2.867% - not the best result. But still good.

I also found max percentage differnece. To do so I look at the smallest and biggest number in the l list. Becasue biggest deviation can be either way. I calculate both percentage differences and compare them. Then print out the one that is the biggest percentage difference with the bionomial result.

$$mx := \frac{\mathrm{abs}(\max(seq(l[i], i = 1..p)) - summ)}{summ} \cdot 100 :$$

$$mn := \frac{\mathrm{abs}(\min(seq(l[i], i = 1..p)) - summ)}{summ} \cdot 100 :$$

if $mx > mn$ **then**

$print(\text{"maximum percentage deviation is"}\ mx);$

else

$print(mn\ \text{"is a maximum percentage deviation"});$

end if:

3.596658043 "is a maximum percentage deviation" **(12)**