Stat3011 Project 2
Title: Statistical Computing Algorithms

Prepared by:
Chan Ho Ying 1155109018
Choi Sen Hei 1155109412
Chui Man Keung 1155110496
Chan King Yeung 1155119394
Lau Tsz Hin 1155110584

Question

1.Implement both bisection and Newton-Raphson algorithms to find the maximal value of $\frac{log(x+x^2)}{1+x^3}$, where x>0

2. Implement a genetic algorithm to maximize the following function with n=10:

$$f(\vec{x}) = \begin{cases} \left| \dfrac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|, & if\ (\forall i,\ 0 \le x_i \le 10)\ and\ (\prod_{i=1}^{n} x_i \ge 0.75) \\ 0, & Otherwise\ (i.e.not\ feasible) \end{cases}$$

3. Implement both a quadrature method and a Monte Carlo method to estimate the integral:
$\int_0^{+\infty} \frac{|cos(x)|}{x} e^{-(log(x)-3)^2}\, dx$

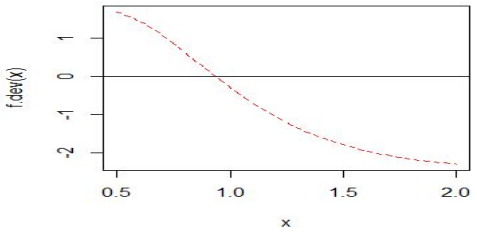
4. Calculate the correlation between x&y of the following distribution:
$p(x,y) \propto sin(\pi x)sin^{20}(\pi x^2) + sin(\pi y)sin^{20}(\pi y^2)\ ,\ 0 \le x,\ y \le 1$

Procedure

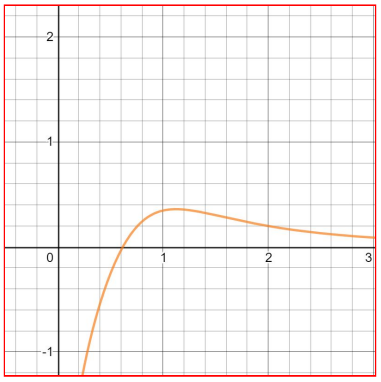
Question 1

1. Bisection

Since we need to find the maximal value of $\frac{log(x+x^2)}{1+x^3}$, so we need to find out the first derivative of $\frac{log(x+x^2)}{1+x^3}$. Also we use the R coding to help us plot the graph.



After that, we can see that the first derivative is positive when x is smaller than 1 while the first derivative is negative when x is greater than 1. It means that the turning point should be between x=0.5 to x=2 for $\frac{log(x+x^2)}{1+x^3}$. Then, we use the R coding to create the bisection method and try to find out the root of first derivative of $\frac{log(x+x^2)}{1+x^3}$. Next we can find out that the root is 1.123657 and we can put x=1.123657 into $\frac{log(x+x^2)}{1+x^3}$. Finally we can find out that the maximal is 0.3595798.



Before using the R coding, we also try the hand calculation. The process is same as the R coding because it is based on the bisection concept. For our finding result, we find out that the root is 1.125 and the maximal is 0.3596.

2. *Newton-Raphson algorithms*

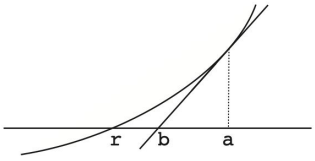
Newton-Raphson algorithms is a root-finding algorithm to produce a approximate value of the roots of a function. The main idea of it is firstly find a starting point which is close to the root, then find the function by the tangent line, and compute the x-intercept of the tangent line.

Let f(x) be the function and r be the root of equation f(x) = 0. Start the estimation with starting point x_0. From x_0, we apply Newton-Raphson algorithms again using the tangent line at $x = x_1$, to produce a new estimate point x_1. We repeat this process to produce new and new estimate points until it becomes 'close enough' to r.

First deriavate $f(x_0)$ to obtain first derivative and second derivative $f'(x_0)$ and $f''(x_0)$.The new estimate point x_1 is obtained from x_0, given by:

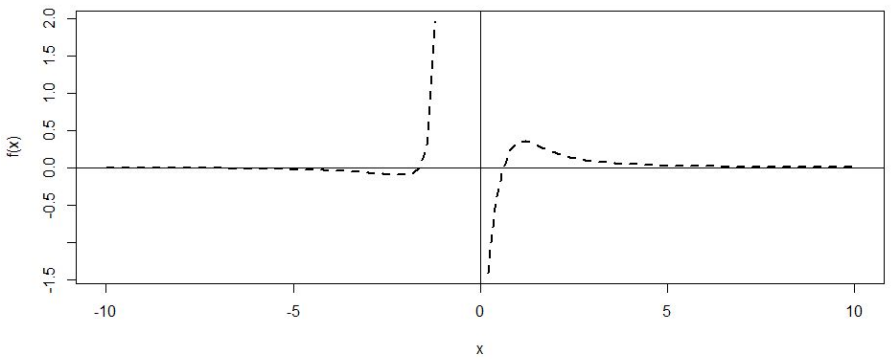
$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$

This process will be repeated until a new point $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ very close to the root is obtained.



In this figure, the curve f(x) reaches the x-axis at point r. b is the point the tangent line touches x-axis. With this figure, this method can be understood easier.

In Question1, f(x) = $\frac{log(x+x^2)}{1+x^3}$, we use R to plot the function:



After that, input f and fdev which is the first and second derivative of f(x). Using the while loop, the process of obtaining a new estimate point which is closer than the previous point is repeated and repeated until it is close enough - the difference between the point and the true root is extremely small. Therefore, we input a variable called "tol", which is e^{-5} , when the difference between them is smaller than "tol", it will quit the while loop and the root will be obtained, which is 1.123312. After substitute x = 1.123312 into f(x), we get 0.3595798.

Before using the R coding, we also try the hand calculation. The process is same as the R coding because it is based on the Newton-Raphson algorithms. For our finding result, we find out that the root is 1.1233 and the maximal is 0.3596.

3. Comparison between bisection and Newton-Raphson algorithms

Newton-Raphson algorithms converge much faster than bisection method because it requires only a single function evaluation per iteration.

On the other hand, bisection takes more time than NR method but since the root is within the bound of interval, the advantage is that it is guaranteed to converge at the end although it takes time. Bisection is also quite difficult to extend to use for systems of equations and some more complicated equations.
To compare the time cost among these two methods, we use a function in R, Sys.time() to record the time used. We will put the start.time right before the loop starts, and end.time right after the root is found. We can get the time used by difference between end.time and start.time. It takes 0.01479292 secs for NR method while 0.02999687 for bisection method.

NR method: `Time difference of 0.01479292 secs`

Bisection method: `Time difference of 0.02999687 secs`

Although the difference is only about 0.015, but the time cost of bisection method is double of NR method, when we generate it 100 times, it costs 1.735267 secs for bisection but only 0.8422348 secs for NR method. If the function is more complicated or the quantity is higher, the time difference will be larger.

To conclude,Newton-Raphson algorithms is relatively more effective than bisection method in terms of its order of convergence. Therefore, NR method is suggested when comparing with bisection and NR method.

Also, from the hand calculation, we can see that even finding the second derivative for Newton-Raphson algorithms is hard, but the time for finding the maximal is quite similar. Besides, we can see that even the maximal is 0.3596, the root of Newton-Raphson algorithms is 1.1233, which is more accurate than Bisection method of root(1.125). It shows that the NR method is suggested when comparing with bisection and NR method.

Question 2

Genetic Algorithm (GA) is a robust methodology to evaluate an optimal point of an objective function. The algorithm makes use of the concept of the theory of natural selection emphasized by Charles Darwin. In general, there are 3 major processes to evaluate a set of solutions, which are selection, crossover, and mutation. For every generation, the population will experience the above procedures to evolve the next generation. Of course, there are some criteria for each step. For those who can adapt the change in the environment has a higher probability to survive into the next generation. They will encounter the process of reproduction and mutation. The pseudocode of using GA is as follows.

1. Start with generating the initial set of solutions (population)
2. Repeat
   Selection, Crossover, and Mutation
3. Obtain the optimal point

Here we will discuss each step in details along with the targeted object function. The first step is to randomly generate an initial set of solutions. Each solution has its own chromosomes. Those are very important to evaluate their fitness of score, which determines whether a solution can go to the next generation and reproduce their offspring. Below are some sample outputs from the source code.

```
> head(initialPopulation)
           Solution1  Solution2 Solution3 Solution4  Solution5 Solution6
Cromosome1 2.5332022 4.18271053  3.744392 3.6043213 5.34458684 0.1825068
Cromosome2 0.8765101 2.12262579  7.452201 1.7445796 4.62805059 3.5074002
Cromosome3 0.1548834 2.77182563  9.744714 4.7971930 4.12785411 4.2825206
Cromosome4 1.0931321 7.81456373  8.398711 8.1533433 0.07370348 9.3171692
Cromosome5 3.9543806 9.01292099  4.638192 9.1407973 0.13496198 9.9777543
Cromosome6 6.5033149 0.04582352  7.098797 0.9079936 7.64217440 0.9032987
```

```
> fitnessEval(initialPopulation)
[1] 0.11336280 0.08461128 0.05361489 0.05909271 0.08738563 0.10118555
```

The purpose of selection is to discover the relative best solution under the population. In other words, a solution with higher fitness of score will have a higher chance to survive among other solutions. Only the selected solutions can have a chance to experience the process of crossover and mutation. For those who are not being selected, they will face the issue of extinction which means they are being dropped from the population. In the selection method, roulette wheel selection is a common approach to select solutions. Each solution is

assigned a probability based on their fitness of score, which is $Pr(solution_i) = \frac{f(solution_i)}{\sum f(solution_i)}$.

GA, then, calculates the cumulative sum of probability for each solution. If a random point ranging in 0 and 1 falls into the slot of the solution, the corresponding solution is being selected. During the process of selection, the solution with the highest fitness of score will be retained for the efficiency of the programme.

After the process of selection, the function of the crossover is being executed. The aim of crossover is to generate a new set of solutions that take the advantages from the previous set of solutions, such that they can inherit the ability to adapt to new change in the environment. The meaning of crossover is to copy partial chromosomes from their parents. Such action will help the child to obtain a higher value of fitness of score. From the right sample outputs, we can see that the new solution gets red colour and green colour from the parents. Indeed, the process of crossover does not necessarily happen, which depends on the probability of crossover. The output of the crossover forms the next generation and the solutions in the generation is expected to have a higher fitness of score than the previous generation.

```
> crossover(x, probCrossover)
           Solution1 Solution2  Solution3 Solution4 Solution5 Solution6 Solution7 Solution8 Solution9 Solution10
Cromosome1  2.793816 9.4378422 4.18271053 3.6043213 2.5332022 9.4378422 9.4378422  3.744392 9.4378422   2.793816
Cromosome2  1.850924 0.1026439 2.12262579 1.7445796 0.8765101 0.1026439 0.1026439  7.452201 0.1026439   1.850924
Cromosome3  9.541761 3.7675276 2.77182563 4.7971930 0.1548834 3.7675276 9.7447139  3.767528 3.7675276   9.541761
Cromosome4  3.869359 6.4624206 7.81456373 8.1533433 6.4624206 1.0931321 8.3987106  6.462421 6.4624206   3.869359
Cromosome5  2.915124 4.1424598 9.01292099 9.1407973 4.1424598 3.9543806 4.6381915  4.142460 4.1424598   2.915124
Cromosome6  9.684185 2.8716899 0.04582352 0.9079936 2.8716899 6.5033149 7.0987968  2.871690 2.8716899   9.684185
Cromosome7  6.085711 2.4774582 7.15283849 7.7543854 2.4774582 9.8368572 9.9495752  2.477458 2.4774582   6.085711
Cromosome8  4.647248 2.4056230 6.57970378 2.7017526 2.4056230 0.2504151 7.2174144  2.405623 2.4056230   4.647248
Cromosome9  7.133505 2.1784833 7.89407557 0.6824263 7.1335054 1.0831396 7.7064728  7.133505 2.1784833   7.133505
Cromosome10 4.112735 5.9615679 8.84394499 8.2571374 4.1127350 7.2606301 9.5519769  4.112735 5.9615679   4.112735
> initialPopulation
           Solution1  Solution2 Solution3 Solution4  Solution5 Solution6 Solution7 Solution8 Solution9 Solution10
Cromosome1 2.5332022 4.18271053  3.744392 3.6043213 5.34458684 0.1825068  8.894970  2.793816  7.217438  9.4378422
Cromosome2 0.8765101 2.12262579  7.452201 1.7445796 4.62805059 3.5074002  7.229408  1.850924  1.672605  0.1026439
Cromosome3 0.1548834 2.77182563  9.744714 4.7971930 4.12785411 4.2825206  9.181520  9.541761  1.525048  3.7675276
Cromosome4 1.0931321 7.81456373  8.398711 8.1533433 0.07370348 9.3171692  3.740069  3.869359  3.333599  6.4624206
Cromosome5 3.9543806 9.01292099  4.638192 9.1407973 0.13496198 9.9777543  9.052893  2.915124  0.700382  4.1424598
Cromosome6 6.5033149 0.04582352  7.098797 0.9079936 7.64217440 0.9032987  5.425081  9.684185  6.824790  2.8716899
Cromosome7 9.8368572 7.75438542  9.949575 7.1528385 9.66870801 0.2399181  7.196991  6.085711  7.627525  2.4774582
Cromosome8 0.2504151 2.70175257  7.217414 6.5797038 8.97387309 1.4544940  9.139718  4.647248  2.921399  2.4056230
Cromosome9 1.0831396 0.68242633  7.706473 7.8940756 1.30074935 7.0827335  5.606471  2.178483  1.967558  7.1335054
Cromosome10 7.2606301 8.25713743  9.551977 8.8439450 1.92124122 7.6296159  3.425094  5.961568  2.516759  4.1127350
>
```

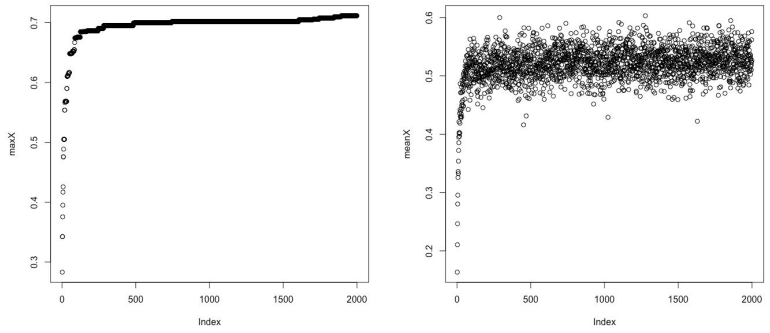
With the defined mutation probability, each solution may experience the mutation. The function of mutation changes some chromosomes of a solution at random. The reason for mutation is to explore any new way to fit the objective function. The mutation plays an important role in GA, which aims to discover the whole search space and avoids the algorithm trap in the local extreme. The following are some outcomes from the GA to illustrate the change in genes of a solution. The colour in red represents the mutation effect to a solution, whereas the colour in green describes the original chromosomes in the solution.

```
> (x = crossover(x, probCrossover))
           Solution1 Solution2 Solution
Cromosome1  4.182711 3.7443925 2.793816
Cromosome2  2.122626 0.1026439 0.102643
Cromosome3  4.797193 9.5417611 3.767527
Cromosome4  8.153343 3.8693590 6.462420
Cromosome5  9.140797 2.9151239 4.142459
Cromosome6  9.684185 0.9079936 2.871689
Cromosome7  2.477458 7.7543854 2.477458
Cromosome8  2.405623 2.4056230 2.701752
Cromosome9  7.133505 7.1335054 2.178483
Cromosome10 7.260630 5.9615679 4.112735
```

```
> mutation(x, probMutatuin)
[1] "Yes"
           Solution1 Solution2 Solution
Cromosome1  2.449859 3.7443925 2.793816
Cromosome2  2.122626 0.1026439 0.102643
Cromosome3  6.589532 9.5417611 3.767527
Cromosome4  6.674933 3.8693590 6.462420
Cromosome5  3.357719 2.9151239 4.142459
Cromosome6  9.684185 0.9079936 2.871689
Cromosome7  2.477458 7.7543854 2.477458
Cromosome8  9.714982 2.4056230 2.701752
Cromosome9  7.133505 7.1335054 2.178483
Cromosome10 7.260630 5.9615679 4.112735
>
```

Up to here, we are all set to perform the GA for the given objective function. We generate 2000 generations to obtain the optimal point for the objective function. The final result we get is 0.71156096, and the corresponding chromosomes are (3.0926655, 3.0256494, 2.9405017, 1.3485606, 0.5738356, 0.4326887, 0.6577955, 0.6539435, 0.5798318, 0.3282071). Below are the graphs describing the maximum and the mean fitness of score in each generation. We can notice that at the end of the graph, the points are not following a strict line. We cannot say the result we have obtained is the optimal point, but with very close to the optimal point as the increase in fitness of score is no longer significant. To get a more precise solution, an increase in the population size and the number of generations does help the GA get an accurate result. However, such an action is indeed a concern to the time complexity. The

longer evaluation time is needed to make a better solution and such action depends on the resources and the decision made by the researchers.

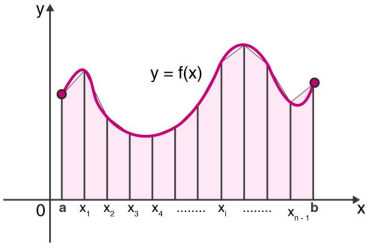


Question 3

1. *Quadrature Method*

Numerical integration methods can be described as combining evaluations of the integrand to get an approximation to the integral. The integrand is evaluated at integration points and a weighted sum of these values is used to approximate the integral. Quadrature is commonly used in numerical integration.Since we are going to find out the integral $\int_0^{+\infty} \frac{|cos(x)|}{x} e^{-(log(x)-3)^2}\, dx$, we applied trapezoidal rule to estimate the integral with R.

For the trapezoidal rule, the integral is divided into small subintervals with a trapezoid shape like the figure shown. The total area of these trapezoids is approximate the the definite integral. So we can calculate the area to estimate the true value of the definite integral.



$$\int_a^b f(x)\,dx \approx \frac{\Delta x}{2} \sum_{k=1}^{N} \left(f(x_{k-1}) + f(x_k)\right)$$

We defined variable f as the integral function $\frac{|cos(x)|}{x} e^{-(log(x)-3)^2}$, variable range as the boundaries of the integral, variable n as the number of subintervals. In the trapezoidalRule function, we defined variable h as Δx and variable i as the subintervals. The function returns total area of the trapezoids as result. The final result is 1.128537.

2. *Monte Carlo Method*

Besides the qudrature method, the integral can be estimated by Monte Carlo integration. Recall the integral $\int_0^{+\infty} \frac{|cos(x)|}{x} e^{-(log(x)-3)^2}\, dx$, there exists a probabilty density function $p(x) = \frac{1}{x\sqrt{\pi}} e^{-(log(x)-3)^2}$, $x > 0$ following by log-normal distribution X~log-Normal(3,½).

$$f(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(lnx-\mu)^2}{2\sigma^2}\right)$$

pdf of log-normal distribution

We can simplify the integral to $\sqrt{\pi}\int_0^{+\infty} |cos(x)|p(x)dx$. As there exists a probabilty density function satisfy $\int_0^{+\infty} p(x)dx = 1$, we can apply the expectation rule on the integral that $\sqrt{\pi}\int_0^{+\infty} |cos(x)|p(x)dx = \sqrt{\pi}E(|cos(x)|)$. Hence, the approximation of the expected value can be drawn by a mean value of the random samples of p(x) $\sqrt{\pi}E(|cos(x)|) \approx \sqrt{\pi}\frac{1}{n}\sum_{i=1}^{n} |cos(x_i)$.

We created the random samples with sample size equal to 100000 following by log-normal distribution with mean = 3 and standard deviation = ½. According to the law of large numbers, the approximation will be more accurate with the large sample size and the concentrated estimators. We used R coding to generate the random samples and find out the final result. The approximation of the integral $\int_0^{+\infty} \frac{|cos(x)|}{x} e^{-(log(x)-3)^2}\, dx$ by Monte Carlo method is 1.128177.

3. Comparison between two methods

The results of qudrature method and Monte Carlo method are close. Quadrature method give out a slightly greater value. The difference between the integral value and the numerical result by trapezoidal rule is shown in the figure below. The trapezoidal rule overestimates when the integrand is concave up and underestimates when the integrand is concave down. However, Monte Carlo Method brings in randomness into computation. The result we get for each time we run the program can be different. They both give out an estimated value of the integral. So we cannot compare the accuracy of two methods directly.

$$\text{error} = -\frac{(b-a)^3}{12N^2} f''(\xi)$$

The computational time required for quadrature method and Monte Carlo method is 0.165 seconds and 0.015 seconds respectively. Quadrature method takes more time as it

needs to calculate the integral part by part. Where Monte Carlo method needs to calculate the mean value of the random samples only. So it takes less time to complete. Therefore, Monte Carlo method is a better method in terms of computational time.

However, Monte Carlo Method brings in randomness into computation. The result we get for each time we run the program can be different. On the other hand, Quadrature method is consistent and it gives the same result no matter how many time we run the program. Therefore, Quadrature method is a better method in terms of certainty of the answer as it gives result without randomness.

Question 4

1. Quadrature Method

It is known that the sum of probability of all outcomes is equal to 1. Therefore we can get $1 = c\int_0^1\int_0^1 \sin(\pi x)\sin^{20}(\pi x^2) + \sin(\pi y)\sin^{20}(2\pi y^2)\,dxdy$. Hence we can rewrite it as $c = \frac{1}{\int_0^1\int_0^1 \sin(\pi x)\sin^{20}(英^2) + \sin(\pi y)\sin^{20}(2\pi y^2)\,dxdy}$. If we want to find c, we have to find $\int_0^1\int_0^1 \sin(\pi x)\sin^{20}(\pi x^2) + \sin(\pi y)\sin^{20}(2\pi y^2)\,dxdy$ first. However, it is very difficult to calculate it directly. So we use trapezoidal rule to compute it. We first divide it into two parts(i.e. $\int_0^1 \sin(\pi x)\sin^{20}(\pi x^2)dx$ and $\int_0^1 \sin(\pi y)\sin^{20}(2\pi y^2)\,dy$).

For the first part, the variable “f” is the function we are going to find the estimate of its integral. The variable “range” is the range for the interval and the variable “n” is the number of interval under range. For the second part, the variable “f” is replaced by “g” for another part of the function and other variables remains the same. After setting those variable we can run the program under trapezoidal rule $\int_a^b f(x)\,dx \approx (b-a)\frac{f(a)+f(b)}{2}$ and add up all result within the interval. Finally we obtain c=4.785692.

The next step is to find E(X),E(X^2),E(Y),E(Y^2). The process is similar for calculating them. We just take E(X) as an example. E(X)= $c\int_0^1\int_0^1 x[sin(\pi x)sin^{20}(\pi x^2)+sin(\pi y)sin^{20}(2\pi y^2)]dydx$ = $c\int_0^1 xsin(\pi x)sin^{20}(\pi x^2)dx + c\int_0^1 xdx\int_0^1 sin(\pi y)sin^{20}(2\pi y^2)dy$. This time we can divide into 3 parts. They are $\int_0^1 x\,\sin(\pi x)\sin^{20}(\pi x^2)\,dx$, $\int_0^1 \sin(\pi y)\sin^{20}(2\pi y^2)\,dy$ and $\int_0^1 x\,dx$.

We can calculate the first part using trapezoidal rule and the third part can be computed directly to its integral. For the second part, it is already done when we find the constant. As a result, we find E(X)=0.5937665, E(X^2)=0.4068357,E(Y)=0.5346438 and E(Y^2)=0..3381641.

The remaining steps are just simple calculations. E(XY)=0.3142051, Var(X)=0.05427708, Var(Y)=0.05232016, Cov(x,y)=-0.003248426 and Corr(x,y)=-0.06095793. Therefore the correlation between x and y is -0.06095793.

2. Monte Carlo Method

For the Monte Carlo Method, we need to create samples first. We set n=100000 which means we sample 100000 observations of x and y respectively from the uniform distribution. We set such a large number for sample because when the sample is large, the result of estimator will be more concentrated, thus the final result will be more accurate.

For calculation part, we also first find E(X)),E(X^2),E(Y),E(Y^2).we use the runif function in r to account for each of the 100000 samples. The mean and variance have to be divided by the mean of f(x,y) since we only know that p(x,y) is proportional to this distribution but the exact density function of it is unknown. So mean of f(x,y) acts as a normalizer for us to estimate it. After calculation, E(X)=0.59478755, E(X^2)=0.3537722, E(Y)=0.53635560 and E(Y^2)=0.2876773.

The final step is to calculate E(XY), Cov(x,y) and Corr(x,y). The samples for calculating E(XY) are also drawn from uniform distribution. E(XY)=0.31575231, Cov(x,y)=-0.00326532 and Corr(x,y)=-0.06135451. Therefore the correlation between x and y is -0.06135451, which is close to the result obtained by Quadrature method.

3. Comparison between two methods

The computational time required for Quadrature Method and Monte Carlo Method is 1.22 seconds and 0.42 seconds respectively. Quadrature Method takes more time as it needs to calculate the integral part by part. We can also see that the number of lines of R-code for it is much more than that for Monte Carlo Method. Therefore, Monte Carlo Method is a better method in terms of computational time.

However, Monte Carlo Method brings in randomness into computation. The result we get for each time we run the program can be different. On the other hand, Quadrature method is consistent and it gives the same result no matter how many time we run the program. Therefore, Quadrature method is a better method in terms of certainty of the answer as it gives result without randomness.

Appendix

Question1

Bisection(Hand Calculation):

Let f(x) = $\frac{ln(x+x^2)}{1+x^3}$, where x>0

Here

x	2	1	0.5	0
f(x)	0.1991	0.3466	0.3589	“undefined”

Let a_1 =2 , b_1 =1, c_1 =0.5 , $x_i = \frac{a_i+b_i}{2}$, $y_i = \frac{b_i+c_i}{2}$
When f(x_1) > f(a_1) , x_1 will be used for a_2 next step.
When f(x_1) > f(b_1) , x_1 will be used for b_2 next step.
When f(y_1) > f(b_1) , y_1 will be used for b_2 next step.
When f(y_1) > f(c_1) , y_1 will be used for c_2 next step.

i	a_i	f(a_i)	x_i	f(x_i)	b_i	f(b_i)	y_i	f(y_i)	c_i	f(c_i)
1	1.5	0.3021	1.25	0.3502	1	0.3465	0.75	0.1913	0.5	-0.2557
2	1.25	0.3502	1.25	0.3502	1.25	0.3502	1	0.3465	0.75	0.1913
3	/	/	/	/	1.25	0.3502	1.125	0.3596	1	0.3465
4	/	/	/	/	1.125	0.3596	1.125	0.3596	1.125	0.3596

Therefore when $x \approx 1.125$, $f(x) \approx 0.3596$ where is the maximal value of f(x).

Bisection(R Code):

```
f.dev <- function(x){
(1+2*x)/((x+x^2)*(1+x^3))- log(x+x^2)*(3*x^2)/(1+x^3)^2
}
curve(f.dev, xlim=c(0.5,2), col='red', lwd=1.5, lty=2)
abline(h=0)

bisection <- function(f.dev, a, b, z=0.001) {
if(f(a)*f(b)>0)
list(fail="fail to find root")
else{
repeat{
if(abs(b-a)<z) break
x<-(a+b)/2
```

```
if(f(a)*f(x)<0) b<-x else a<-x
}
list(root=(a+b)/2,func=f(x))
}
}
f <- function(x){
(1+2*x)/((x+x^2)*(1+x^3))- log(x+x^2)*(3*x^2)/(1+x^3)^2
}
bisection(f,0.5,2)
$root
[1] 1.123657

x<-1.123657
log(x+x^2)/(1+x^3)
[1] 0.3595798
```

Newton-Raphson algorithms(Hand Calculation)**:**

Let f(x) = $\frac{ln(x+x^2)}{1+x^3}$, where x>0

x	1.5	1	0.5	0
f(x)	0.3021	0.3466	0.3589	"undefined"

$\frac{d}{dx}(\frac{ln(x+x^2)}{1+x^3})$
$= \frac{(2x+1)(x^2-x+1)-3x^3 ln(x+x^2)}{x(1+x^3)^2}$
$\frac{d^2}{dx^2}(\frac{ln(x+x^2)}{1+x^3})$
$= \frac{12x^9 ln(x^2+x)-14x^9+8x^8-9x^7+6x^6 ln(x^2+x)-20x^6+7x^5-9x^4-7x^3-6x^3 ln(x^2+x)-x^2-1}{x^2(1+x^3)^4}$

Therefore
$f'(x) = \frac{(2x+1)(x^2-x+1)-3x^3 ln(x+x^2)}{x(1+x^3)^2}$
$f''(x) = \frac{12x^9 ln(x^2+x)-14x^9+8x^8-9x^7+6x^6 ln(x^2+x)-20x^6+7x^5-9x^4-7x^3-6x^3 ln(x^2+x)-x^2-1}{x^2(1+x^3)^4}$

Let $a = 1.5$ $b = 1$, $c = 0.5$, $x_0 = \frac{a+b}{2}$, $y_0 = \frac{b+c}{2}$
For $x_i = x_{i-1} - \frac{f'(x_{i-1})}{f''(x_{i-1})}$

i	x_i	$f(x_i)$	$f'(x_i)$	$f''(x_i)$	$dx\ (x_{i-1} - x_i)$
0	1.25	0.3502	-0.1344	-0.7462	-0.1802
1	1.0698	0.3573	0.0858	-1.7939	0.0479
2	1.1176	0.3596	0.0082	-1.4593	0.0056

3	1.1232	0.3596	0.00016	-1.4228	0
4	1.1232	0.3596	0.00016	-1.4228	/

For this table, when $x \approx 1.1232$, $f(x) \approx 0.3596$, where is the maximal value of f(x).

i	y_i	$f(y_i)$	$f'(y_i)$	$f''(y_i)$	$dy\ (y_{i-1} - y_i)$
0	0.75	0.1913	1.1126	-4.7262	0.2354
1	0.9854	0.3430	0.2655	-2.4825	0.1069
2	1.0923	0.3589	0.0473	-1.6312	0.0290
3	1.1213	0.3596	0.0029	-1.4351	0.0003
4	1.1233	0.3596	0.000008	-0.6176	0
5	1.1233	0.3596	0.000008	-0.6176	/

For this table, when $x \approx 1.1233$, $f(x) \approx 0.3596$ where is the maximal value of f(x).

Newton Raphson(R code)**:**

```
func <- function(x) {
  (log(x+x^2))/(1+x^3))
}

curve(func, xlim=c(-10,10), col='black', lwd=2, lty=2, ylab='f(x)')
abline(h=0)
abline(v=0)
f <- function(x){
  (2 * x + 1 ) / ((x^2 + x) * (x^3 + 1)) - ((3 * x^2 * log(x^2 + x)) / (x^3 + 1)^2)
}
#first derivative f'(x)
fdev <- function(x){
  (2 / (x + x^2) - (1 + 2*x) * (1 + 2 * x) / ( x + x^2)^2) / (1 + x^3) - (1 + 2 * x) / (x + x^2) * (3
* x^2) / (1 + x^3)^2 - (((1 + 2 * x) / (x + x^2) * (3 * x^2) + log(x + x^2) * (3 * (2 * x))) / (1 +
x^3)^2 - log(x + x^2) * (3 * x^2) * (2 * (3 * x^2 * (1 + x^3))) / ((1 + x^3)^2)^2)
}
x<-0.5
tol<-1e-5
root<- function(f, f.dev, x, tol){
  y=x
  while(abs(f(y)) > tol){
    y = y - f(y)/f.dev(y)
  }
  return(y)
}
```

root(f, fdev, x, tol)

```
> root(f, fdev, x, tol)
[1] 1.123312

> x<-1.123312
> log(x + x^2) / (1 + x^3)
[1] 0.3595798
```

Question 2

```
##############################################################################
#####

# define basic parameters
n <- 10
m <- 100
lb <- 0
ub <- 10
probCrossover <- 0.9
probMutatuin <- 0.5
set.seed(3011)

##############################################################################
#####

# objective function
f <- function(x) {
  # throw an exception: product of x >= 0.75
  if(prod(x) < 0.75)
    return(0)
  else
    return(abs((sum(cos(x) ^ 4) - 2 * prod(cos(x) ^ 2)) / sqrt(sum(x ^ 2 * 1:n))))
}

##############################################################################
#####

compare <- function(){
  return(c(max(fitnessEval(initialPopulation)), max(fitnessEval(x)),
mean(fitnessEval(initialPopulation)), mean(fitnessEval(x))))
}

##############################################################################
#####

# original population
initialPopulation <- matrix(runif(n * m, lb, ub), nr = n, nc = m)
```

```
rownames(initialPopulation) <- paste0("Cromosome", 1:n)
colnames(initialPopulation) <- paste0("Solution", 1:m)

################################################################################
#####

# evaluate the fitness score
fitnessEval <- function(population) {

  fitness <- c()

  for(i in 1:ncol(population)) {
    fitness <- c(fitness, f(population[, i]))
  }

  return(fitness)
}

################################################################################
#####

# sort fitness
fitnessSort <- function(population) {

  fitness <- fitnessEval(population)
  return(population[, order(fitness)])

}

################################################################################
#####

# selection from population
selection <- function(population) {
  population <- fitnessSort(population)
  fitness <- fitnessEval(population)
  probSelect <- fitness / sum(fitness)
  cumProbSelect <- cumsum(probSelect)

  bestSol <- population[, (m * 0.8):m]

  ran <- runif(m * 0.8, 0, 1)
  index <- c()

  for(i in 1:(m * 0.8)) {
    for(j in 1:m) {
      if(ran[i] <= cumProbSelect[j]) {
        index <- c(index, j)
```

```
      break
    }
   }
  }

  newSelectionPopulation <- bestSol
  for(i in 1:(m * 0.8 - 1)) {
   newSelectionPopulation <- cbind(newSelectionPopulation, population[, index[i]])
  }

  newSelectionPopulation <- fitnessSort(newSelectionPopulation)

  rankSelect <- fitnessSort(newSelectionPopulation[, sample(1:m, replace = T, prob =
probSelect)])

  newSelectionPopulation <- cbind(rankSelect[, 1:(m - 1)], newSelectionPopulation[, m])

  colnames(newSelectionPopulation) <- paste0("Solution", 1:m)

  return(newSelectionPopulation)
}

x=selection(initialPopulation)
compare()

################################################################################
#####

# crossover from population
crossover <- function(population, probCrossover) {
  sexPartner <- sample(1:m)
  population <- population[, sexPartner]
  newCrossoverPopulation <- c()

  for(i in seq(1, m, 2)) {
   flag <- sample(c(0,1), 1, prob = c(1 - probCrossover, probCrossover))
   if(flag == 1)
    newCrossoverPopulation <- cbind(newCrossoverPopulation, crossoverADT(population[,
i:(i + 1)]))
   else
    newCrossoverPopulation <- cbind(newCrossoverPopulation, population[, i:(i + 1)])
  }

  # survive solutions from previous generation
  surviveSol <- fitnessSort(population)[, (m * 0.9 + 1):m]
  newCrossoverPopulation <- cbind(fitnessSort(newCrossoverPopulation)[, (m * 0.1 + 1):m],
surviveSol)
```

```
  colnames(newCrossoverPopulation) <- paste0("Solution", 1:m)
  rownames(newCrossoverPopulation) <- paste0("Cromosome", 1:n)
  return(newCrossoverPopulation)
}

crossoverADT <- function(population) {

  iter <- 0

  repeat {
    index <- sample(1:(n - 1), 1)

    sol1_vector <- population[1:index, 1]
    sol1_vector2 <- population[(index + 1):n, 1]

    sol2_vector <- population[1:index, 2]
    sol2_vector2 <- population[(index + 1):n, 2]

    newSol <- c(sol1_vector, sol2_vector2)
    newSol2 <- c(sol2_vector, sol1_vector2)

    if(all(prod(newSol) >= 0.75 & prod(newSol2) >= 0.75))
      break
    else
      iter = iter + 1
    if(iter > 20)
      return(population)
  }
  return(cbind(newSol, newSol2))
}

x = crossover(x, probCrossover)
compare()

##########################################################################
#####

# mutation of population
mutation <- function(population, probMutatuin) {

  newMutationPopulation <- fitnessSort(population)
  for(i in 1:(m - 1)) {
    #i=1
    flag <- sample(c(0,1), 1, prob = c(1 - probMutatuin, probMutatuin))
    if(flag == 1) {
      post <- sample(1:n,1)
      repeat {
        for(j in 1:post) {
```

```
        index <- sample(1:n, 1)
        newMutationPopulation[index, i] <- runif(1, lb, ub)
       }
       if(prod(newMutationPopulation[, i]) >= 0.75)
        break
     }
    }
  }

  # survive solutions from previous generation
  surviveSol <- fitnessSort(population)[, (m * 0.9 + 1):m]
  newMutationPopulation <- cbind(fitnessSort(newMutationPopulation)[, (m * 0.1 + 1):m],
surviveSol)

  return(newMutationPopulation)
}

x = mutation(x, probMutatuin)
compare()

##############################################################################
#####

x <- mutation(crossover(selection(initialPopulation), probCrossover), probMutatuin)
maxX <- c()
meanX <- c()
for(i in 1:2000){
  x <- mutation(crossover(selection(x), probCrossover), probMutatuin)
  maxX <- c(maxX,max(fitnessEval(x)))
  meanX <- c(meanX, mean(fitnessEval(x)))
  print(paste0("Loop", i," Finish"))
}

compare()

plot(maxX)
plot(meanX)

##############################################################################
#####
```

Question 3

Quadrature method

```
start_time<-Sys.time()
f<-function(x){return(abs(cos(x))/x*exp(-(log(x)-3)^2))}
trapezoidalRule<-function(f,range,n){
  h<-diff(range)/n
  i<-range[1]+h*(0:n)
  return(h/2*(f(range[1]+f(range[2]))+2*sum(f(i[-1][-(n+1)]))))}
trapezoidalRule(f,range=c(0,1000),n=1e6)
end_time<-Sys.time()
end_time-start_time
```

Monte Carlo method

```
set.seed(0)
start_time<-Sys.time()
f<-function(x){abs(cos(x))*sqrt(pi)}
p<-function(x){1/(sqrt(pi)*x)*exp(-(log(x)-3)^2)}#pdf
x<-rlnorm(100000,3,sqrt(1/2))
result<-mean(f(x))
result
end_time<-Sys.time()
end_time-start_time
```

Question 4

Quadrature method

```
start_time<-Sys.time()
f<-function(x) {
  return(sin(pi*x)*((sin(pi*x^(2)))^(20)))
  }

T1<-function(f,range,n){
  h<-diff(range)/n
  i<-1:n-1
  s<-0+i*h
  est<-(h/2)*(f(0)+2*sum(f(s))+f(1))
  return(est)}

T1(f,range=c(0,1),100000)                    #find estimate for first part

g<-function(y) {
  return(sin(pi*y)*((sin(2*pi*y^(2)))^(20)))
  }

T2<-function(g,range,n){
```

```
  h<-diff(range)/n
  i<-1:n-1
  s<-0+i*h
  est<-(h/2)*(g(0)+2*sum(g(s))+g(1))
  return(est)}

T2(g,range=c(0,1),100000)                              #find estimate for second part

c<-1/(T1(f,range=c(0,1),100000)+T2(g,range=c(0,1),100000))       #find constant

c

EX1<-function(x) {
  return(x*sin(pi*x)*((sin(pi*x^(2)))^(20)))
  }

T<-function(EX1,range,n){
  h<-diff(range)/n
  i<-1:n-1
  s<-0+i*h
  est<-(h/2)*(EX1(0)+2*sum(EX1(s))+EX1(1))
  return(est)}

T(EX1,range=c(0,1),100000)   #find estimate for first part

T2(g,range=c(0,1),100000)    #find estimate for second part

x<-function(x){x}            #find estimate for third part
integrate(x,0,1)

EX<-c*(T(EX1,range=c(0,1),100000)+T2(g,range=c(0,1),100000)*0.5)   #find E(X)
EX

EX2<-function(x) {
  return((x^2)*sin(pi*x)*((sin(pi*x^(2)))^(20)))
}

T<-function(EX1,range,n){
  h<-diff(range)/n
  i<-1:n-1
  s<-0+i*h
  est<-(h/2)*(EX2(0)+2*sum(EX2(s))+EX2(1))
  return(est)}

T(EX2,range=c(0,1),100000)   #find estimate for first part

T2(g,range=c(0,1),100000)    #find estimate for second part
```

```
x2<-function(x){x^2}        #find estimate for third part
integrate(x2,0,1)

EX2<-c*(T(EX2,range=c(0,1),100000)+T2(g,range=c(0,1),100000)*1/3)   #find E(X)^2
EX2

T1(f,range=c(0,1),100000)    #find estimate for first part

EY1<-function(y) {
  return(y*sin(pi*y)*((sin(2*pi*y^(2)))^(20)))
}

T<-function(EY1,range,n){
  h<-diff(range)/n
  i<-1:n-1
  s<-0+i*h
  est<-(h/2)*(EY1(0)+2*sum(EY1(s))+EY1(1))
  return(est)}

T(EY1,range=c(0,1),100000)  #find estimate for second part

y<-function(y){y}           #find estimate for third part
integrate(y,0,1)

EY<-c*(T(EY1,range=c(0,1),100000)+T1(f,range=c(0,1),100000)*0.5)  #find E(Y)
EY

T1(f,range=c(0,1),100000)    #find estimate for first part

EY2<-function(y) {
  return((y^2)*sin(pi*y)*((sin(2*pi*y^(2)))^(20)))
}

T<-function(EY2,range,n){
  h<-diff(range)/n
  i<-1:n-1
  s<-0+i*h
  est<-(h/2)*(EY2(0)+2*sum(EY2(s))+EY2(1))
  return(est)}

T(EY2,range=c(0,1),100000)  #find estimate for second part

y2<-function(y){y^2}        #find estimate for third part
integrate(y2,0,1)

EY2<-c*(T(EY2,range=c(0,1),100000)+T1(f,range=c(0,1),100000)*1/3)  #find E(Y)^2
EY2
```

```
EXY1<-(T(EX1,range=c(0,1),100000)*0.5)  #find estimate for first part
EXY1

EXY2<-(T(EY1,range=c(0,1),100000)*0.5)  #find estimate for second part
EXY2

EXY<-c*(EXY1+EXY2) #find E(XY)
EXY

VarX<-EX2-EX^2     #find Var(X)
VarX

VarY<-EY2-EY^2     #find Var(Y)
VarY

cov<-EXY-EX*EY     #find COV(X,Y)
cov

Corr<-cov/sqrt(VarX*VarY) #find Corr(X,Y)
Corr
end_time<-Sys.time()
end_time-start_time
```

Monte Carlo method

```
start_time<-Sys.time()
f<-function(x,y){
  sin(pi*x)*(sin(pi*x^2))^(20)+sin(pi*y)*(sin(2*pi*y^2))^(20)
}
n=100000
x=runif(n)
y=runif(n)
EX=mean(x*f(x,y))/mean(f(x,y))
EX^2
varX=mean(x^2*f(x,y))/mean(f(x,y))-EX^2
EY=mean(y*f(x,y))/mean(f(x,y))
EY^2
varY=mean(y^2*f(x,y))/mean(f(x,y))-EY^2
EXY=mean(x*y*f(x,y))/mean(f(x,y))
covXY=EXY-EX*EY
corr=covXY/sqrt(varX*varY)
end_time<-Sys.time()
end_time-start_time
```