OPTIMIZATION OF BINARY RANDOMIZED RESPONSE BY USING NORMAL DISTRIBUTION FILTER

BY

MR. PEERAKARN THONGSATA
MR. TITHIVICH PAKA
MR. TOUCH SUNGKAWICHAI

A REPORT SUBMITED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR GRADUATION IN 2020

KAMNOETVIDYA SCIENCE ACADEMY (KVIS)

Project Title	OPTIMIZATION OF BINARY RANDOMIZED RESPONSE BY USING NORMAL DISTRIBUTION FILTER
By	Mister Peerakarn Thongsata
	Mister Tithivich Paka
	Mister Touch Sungkawichai
Field of Study	Mathematics
Project Advisor	Mr. Atiratch Laoharenoo
Academic year	2020

ABSTRACT

Anonymous polls nowadays rely on the owner's company not to share responses or look through the responses. Realizing this, the randomized responses technique was developed to make polling more secure considering the privacy of the client. In this paper, we proposed a method of applying a normal random variable to optimize the sampling technique in terms of privacy and accuracy.

Keywords: Randomized Response, Polling, Secure polling, Population Sampling Technique, Indirect Population Sampling

ACKNOWLEDGEMENTS

We are thankful for our advisor's comments and helps.

Mr. Peerakarn Thongsata
Mr. Tithivich Paka
Mr. Touch Sungkawichai

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Importance

Especially in the age of data, gathering information from people is beneficial to companies, organizations, and institutes. People might feel like it is safer to enter private information to a secured, trusted organizations via their anonymous surveys, while in reality, the survey owner has full control and access to networking details including the IP address. So it's trivial for the owner to decode back and find the corresponding answer of every individual. In frank, anonymous polls nowadays rely fully on the trustworthiness of the obligated organization and their policies.

1.2 Research Objective

The goal of this research is to improve the security of the polling method while increasing a relatively small amount of error. The expected result is an anonymous surveying model that guarantees high privacy from everyone including the survey owner company, and the error must lie in the acceptable rate.

CHAPTER 2
REVIEW OF LITERATURE

2.1 Preliminary

2.1.1 Likelihood Function

The likelihood function is the function that measures how well a static model fits sample data. The likelihood function describes a hypersurface, where its maximum if exists, represents the combination of model parameter values that maximize the probability of drawing the sample obtained. The procedure for obtaining these arguments of the maximum of the likelihood function is known as maximum likelihood estimation.

Let $X_1, X_2, \ldots, X_n$ be observations from n independent and identically distributed random variables drawn from a Probability Distribution f that depend on some parameter θ on parameter space θ.Then the likelihood function is

$$L = f(X_1, X_2, \ldots, X_n|\theta) = f(X_1|\theta) \times f(X_2|\theta) \times \cdots \times f(X_n|\theta)$$

where $\theta \in \Theta$.

2.1.2 Maximum Likelihood Estimation

Maximum likelihood estimation (MLE) (Le Cam) is a method of estimating the parameters of a probability distribution by maximizing a likelihood function so that under the assumed statistical model the observed data is most probable. The point in the parameter space that maximizes the likelihood function is called the maximum likelihood estimate.

The goal of maximum likelihood estimation is to find the values of the model parameters that maximize the likelihood function over the parameter space Θ, that is

$$\hat{\theta} = \arg_{\theta \in \Theta} \max L$$

The specific value $\hat{\theta} \in \theta$ is called the maximum likelihood estimate. If it is measurable, then it is called the maximum likelihood estimator. Thus, probability density function of random variable X can be shown as

$$N(X; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

2.1.3 Normal Random Variable

A "Normal random variable" is called the random variable X, distributed in an inverted bell shape with 2 parameters; mean μ and variance σ^2.

2.1.4 Baye's Theorem

Bayes' theorem (Bayes) describes how to update the probabilities of hypotheses when given evidence. It follows simply from the axioms of conditional probability but can be used to powerfully reason a wide range of problems involving belief updates.

Given an experiment, the universe U include n unsimultaneous events $A_1, A_2, \ldots, A_n$, and E be and event in the sample space given by $A_i (i = 1, 2, \ldots, n)$. The conditional probability of A_i given by E which was already occur will be able to determine by the equation

$$P(A_i|E) = \frac{P(E|A_i) \cdot P(A_i)}{\sum_{i=1}^{n} P(E|A_i) \cdot P(A_i)} = \frac{P(E|A_i) \cdot P(A_i)}{P(E)}$$

2.2 Literature Review

2.2.1 Randomized Response

Randomized response Warner is created to prevent the situation where a survey is made in sensitive topics, as anonymous voting is not absolutely anonymous - the obligated organization can look up the voters' IP address and backtrack for their personal information. Randomized response helps in a collection of opinions on a yes-no question on sensitive topics that require an additional level of privacy. The method involves asking the respondents to flip a coin. If it is head then they will report their answer. If it's tail, they will flip again and report the result of the second coin flip. The report will be 'yes' if the coin land in the head, and the other way round. Certainly, the respondents must keep the coin flip information secret.

With the mentioned method, it's easy to statistically decode the total number of voters on each side, while ensuring the confidentiality that no one can decode back the individual response. Randomized response works well with a large sample, but not with a small population.

2.2.2 RAPPOR

RAPPOR, short for Randomized Aggregatable Privacy-Preserving Ordinal Response, is a crowdsourcing method that ensures anonymity for end-users. (Úlfar Erlingsson) It involves using the idea of randomized responses (2.2.1) and using 2 levels of filter, the classical randomized response filter called the permanent randomized response, and another one called the instantaneous response. In the binary survey, when a user opts to vote on either choice, they when pass their choice into the permanent randomized response and keep the value forever. However, when they want to make a response, they will pass their permanent value into another layer (instantaneous filter) and submit the result. This way they can submit multiple times and still guarantee enough privacy. This is the way RAPPOR prevents longitudinal attacks. In ad-

dition, RAPPOR also supports more than a binary survey, but also any ordinal survey. This work by passing the number from the client into a bloom filter, then pass the bloom filter into two mentioned layers of randomized response generator.

CHAPTER 3
RESEARCH METHODOLOGY

3.1 Study of the Randomized Response algorithm

Randomized Response is a method to prevent tracking of how the individual vote contributes to the poll result. This is done by altering the chosen vote to the opposite choice with a one-quarter probability. The goal of this research is to increase the privacy of the model while increasing a little amount of error relative to the benefits.

3.2 Simulate poll results using the optimized Randomized Response algorithm

A program is written to compute the poll results after applying the original Randomized Responses algorithm with various values of numbers of total votes and the probability of changing the vote. Set of mean and standard deviation are determined as fixed values and write another program to compute the poll results from the optimized Randomized Responses algorithm with the same set of numbers of total votes and probability of changing the vote. Repeating both programs many times to find the root mean square deviation.

3.3 Analyze the poll results

Privacy function and accuracy function were defined to measure the efficiency and correctness of the Randomized Responses algorithm. Both functions were used to plot error graphs and accuracy graphs of the poll results from the original Randomized Response algorithm and the optimized Randomized Response algorithm with various values of mean and standard deviation. Then compare and analyze the differences in both graphs.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Defining Functions

For the sake of calculation, let assume without loss of generality that the total number of voters is N, and among N people, there are n people that intend to vote 1 (choose $v = 1$). Furthermore, let define $p = \frac{n}{N}$ as the ratio of people that intend to vote 1.

4.1.1 Classical Filter

Firstly, when users select to vote either choice, say $v \in \{0,1\}$, they will need to make a randomized response out of the vote they of their choice. The usual definition of randomized response involves fair coin flips. The first coin flip determines whether the user should report truthfully or report randomly. The second coin flip is then used to randomize the report when they need to report randomly.

To generalize this, let define it to be a random function that takes v, the voters' intended choice, and returns the randomized vote.

$$R_c(v) = \begin{cases} v & \text{with probability of } f, \\ 1 & \text{with probability of } (1-f)q, \\ 0 & \text{with probability of } (1-f)(1-q) \end{cases} . \tag{1}$$

It's easy to see that $R_c(v) \in \{0,1\}$. When user opted to vote v, they will vote $R_c(v)$ instead. The likelihood function (2.1.2) is then defined as

$$L(p) = [pf + (1-f)q]^c \cdot [(1-p)f + (1-f)(1-q)]^{N-c}$$

where c is number of $\sum_{i=1}^{N} R_c(v_i)$. The maximum value of the likelihood function is found when $p = \frac{c-(f-1)qN}{fN}$. From this, we get that the point estimator of n.

$$\hat{n} = \frac{c+(1-f)qN}{f} \tag{2}$$

4.1.2 Normal Random Variable Filter

Apart from the classical filter, the following is the definition of Normal Random Variable Filter.

$$R_n(v) = \begin{cases} f_N(\sigma, 0) & \text{if } v = 1 \\ f_N(\sigma, \delta) & \text{if } v = 0 \end{cases} , \tag{3}$$

where $f_N(\sigma, \mu)$ is a Normal random variable which distribution probability density function is with parameter σ and μ. Please note that $R_n(v) \in \mathbb{R}$.

Using maximum likelihood estimation (2.1.2), it's possible to find the point estimator for p, which is $\hat{p} = 1 - \frac{c}{N\delta}$. Consequently, the following is the point estimator for n.

$$\hat{n} = \frac{N\delta - c}{\delta}, \tag{4}$$

where c is number of $\sum_{i=1}^{N} R_n(v_i)$.

4.2 Privacy

4.2.1 Classical filter

From the classical model, we can find the following probability equations.

$$\begin{aligned}
P(v=1) &= p \\
P(v=0) &= 1-p \\
P(R_c(v)=1) &= fp+(1-f)q \\
P(R_c(v)=0) &= f(1-p)+(1-f)(1-q) \\
P(R_c(v)=1|v=1) &= f+(1-f)q \\
P(R_c(v)=0|v=1) &= (1-f)(1-q) \\
P(R_c(v)=1|v=0) &= (1-f)q \\
P(R_c(v)=0|v=0) &= f+(1-f)(1-q)
\end{aligned}$$

We further define Insecurity of the model as the ability to guess back the value of v when the value $R(v)$ is known, when determining over every value of possible $R(v)$ and p.

$$I(v) = \int_{-\infty}^{\infty} [|P(v=1|R(v)=x) - P(v=0|R(v)=x)| \cdot P(R(v)=x)]dx \tag{5}$$

which gives

$$\begin{aligned}
I(v) &= |P(R_c(v)=0|v=1)\cdot P(v=1) - P(R_c(v)=0|v=0)\cdot P(v=0)| \\
&\quad + |P(R_c(v)=1|v=1)\cdot P(v=1) - P(R_c(v)=1|v=0)\cdot P(v=0)| \\
&= |(2p-1)(1-f)(1-q)+pf| \ + \ |(2p-1)(1-f)q+pf-f|
\end{aligned}$$

4.2.2 Normal Filter

Similarly, the definition of Insecurity of the Normal Random Filter model resembles the Insecurity of the classical model (5). This model, which inherit from the classical filter model, can be analyzed in the following way.

$$I(v) = \int_{-\infty}^{\infty} (|f(x;\sigma,0)\cdot p| \ + \ |f(x;\sigma,\delta)\cdot(1-p)|)\ dx.$$

where f is the pdf of the normal random variable, which is derived into

$$I(v) = (p-1)\,\mathrm{erf}\left(\frac{I_{ntersect}-\delta}{\sigma\sqrt{2}}\right) - p\,\mathrm{erf}\left(\frac{-I_{ntersect}}{\left(\sigma\sqrt{2}\right)}\right)$$

where $I_{ntersect}$ is the the intersection point of two normal pdf, which is $\frac{\delta^2 \ - \ 2\sigma^2\left(\ln\left(\frac{1}{p}-1\right)\right)}{2\delta}$

4.3 Accuracy

Both classical and the random normal filter model are simulated to see how well each model perform. The simulation of the classical model can be formally written as an object of four parameters: p - the population bias (the independent probability that each person is going to vote for 1), f - truth probability (the probability that they will report their real v), q - vote 1 probability (the probability that they will report 1 if they do not report their own opinion), and N - the size of the population.

In the same way, the simulation of the normal random variable can be written as an object of four parameters: p, N similar to the classical model, σ and δ for the normal random variable.

The error is measured by finding the difference between the number of people who actually vote 1 and the expected result from the model (from the equation 2). A simulation is run in 100 epochs using the root mean square (RMS) method to emphasize the outliers.

The code for the simulation is attached in Appendix A.

From the result, it can be seen that the Normal Random Variable can achieve a similar result to the classical way. The Insecurity and Error depend on the σ and δ for the Normal Random Variable, and the f and q for the classical method.

From the figures in Appendix B, we see the overall value of both insecurity and error. The normal variation of this model has only one peak in both aspects, and the classical method has two peaks in both aspects. It's known that the lower the value, the better the model, so these figures show the normal random variable filter can help optimize the randomized responses technique.

4.4 Integration with RAPPOR

The proposed combined filter model resembles RAPPOR (2.2.2) Algorithm. It's possible for the poll organizer to set up a survey that protects longitudinal attack, and ensure privacy even the users submit repeatedly infinitely many times using the RAPPOR.

CHAPTER 5
CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

A normal random variable can be used to optimize for a better model of randomized response technique, which works even for a small population. It's possible with this optimization to hope for an increment in privacy and also the accuracy of this statistical model.

5.2 Recommendations

This research is very limited to the special case of normal variable filter, so it can be improved in many aspects.
- switching for a random generator other than normal random variable
- a dynamic q random model
- a non-symmetric random generator
- using a pair of normal variables with a different standard deviation
- generalize the survey model to a string-based response (the way it is done in RAPPOR)

REFERENCES

URL: https://online.stat.psu.edu/stat414/lesson/26/26.1.

Bayes, T. "An essay towards solving a problem in the doctrine of chances". In: *Phil. Trans. of the Royal Soc. of London* 53 (1763), pp. 370–418.

Le Cam, L. *Maximum Likelihood; An Introduction**. URL: https://www.stat.berkeley.edu/~rice/LeCam/papers/tech168.pdf.

Úlfar Erlingsson Vasyl Pihur, Aleksandra Korolova. "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response". In: (). URL: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/42852.pdf.

Warner, Stanley L. "Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias". In: *Journal of the American Statistical Association* 60.309 (1965). PMID: 12261830, pp. 63–69. DOI: 10.1080/01621459.1965.10480775. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1965.10480775. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1965.10480775.

APPENDIX A

```
from random import random
import numpy as np
import matplotlib.pyplot as plot
from matplotlib import cm
from datetime import datetime
import pytz


def withProb(p):
    # ASSUME THAT THIS WORKS
    return random() < p


def normalize(n, N, sigma, delta):
    cnt = 0
    for i in range(n):
        cnt += np.random.normal(0, sigma)
    for i in range(N - n):
        cnt += np.random.normal(delta, sigma)
    return cnt


def denormalize(n, N, sigma, delta):
    return (n - N * delta) / (-1 * delta)


def noisify(n, N, f, q):
    bloom = lambda v: v if withProb(f) else 1 if withProb(q)
       else 0
    cnt = 0
    for i in range(n):
        cnt += bloom(1)
    for i in range(N - n):
```

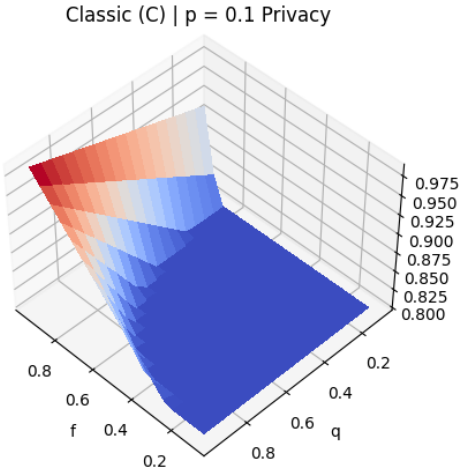
```
        cnt += bloom(0)
    return int(cnt)


def denoisify(n, N, f, q):
    return (n - (1 - f) * N * q) / f


Ns = [100, 500, 1000, 1500, 2000, 2500, 3000, 3500]
ps = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
epoch = 100
fs = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
qs = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
sigmas = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
deltas = [0.2, 0.4, 0.6, 0.8, 1]

print("sigmas: ", sigmas)
print("deltas: ", deltas)
print("fs: ", fs)
print("qs: ", qs)
print("Ns: ", Ns)
print("ps: ", ps)

Error_s_normal = []
for sigma in sigmas:
    Error_d_normal = []
    for delta in deltas:
        Error_normal = []
        for N in Ns:
            error_P_normal = []
            for p in ps:

                n = int(p * N)

                error_normal = 0
```

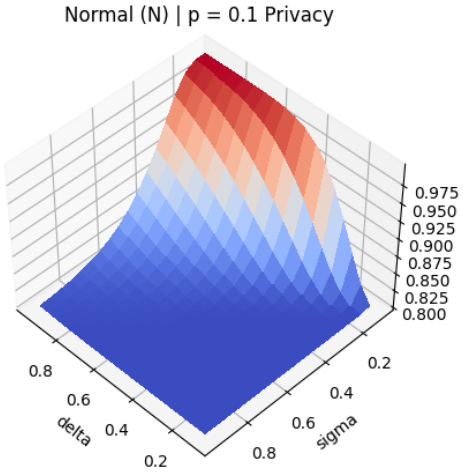
```
                for i in range(epoch):

                    noisy = normalize(n, N, sigma, delta)
                    noisy = denormalize(noisy, N, sigma, delta
                        )

                    error_normal += (noisy - n) * (noisy - n)

                error_normal = pow(error_normal / epoch, 0.5)
                error_P_normal.append(error_normal)
            Error_normal.append(error_P_normal)
        Error_d_normal.append(Error_normal)
    Error_s_normal.append(Error_d_normal)

print("Error_s_normal:␣", Error_s_normal)

Error_q_normal = []
for q in qs:
    Error_f_normal = []
    for f in fs:
        Error_normal = []
        for N in Ns:
            error_P_normal = []
            for p in ps:

                n = int(p * N)

                error_normal = 0

                for i in range(epoch):

                    noisy = noisify(n, N, f, q)
                    noisy = denoisify(noisy, N, f, q)
```

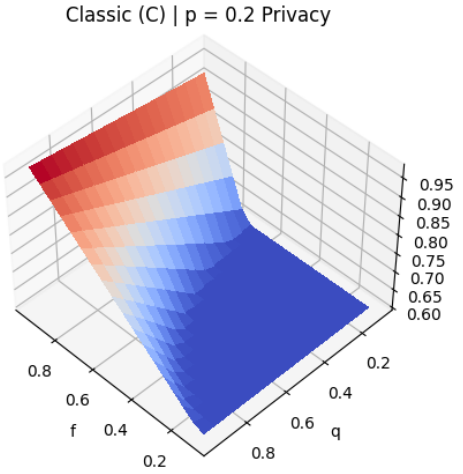
```
                    error_normal += (noisy - n) * (noisy - n)

                error_normal = pow(error_normal / epoch, 0.5)
                error_P_normal.append(error_normal)
            Error_normal.append(error_P_normal)
        Error_f_normal.append(Error_normal)
    Error_q_normal.append(Error_f_normal)


print("Error_q_normal: ", Error_q_normal)
```

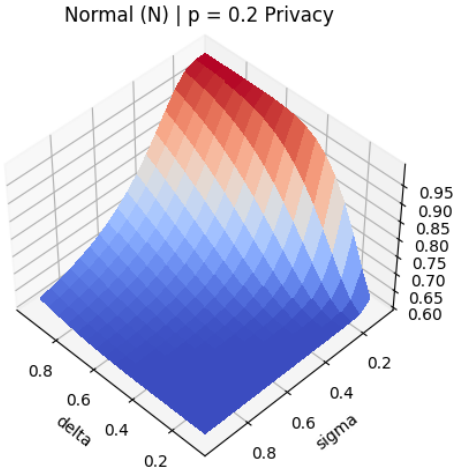
APPENDIX B

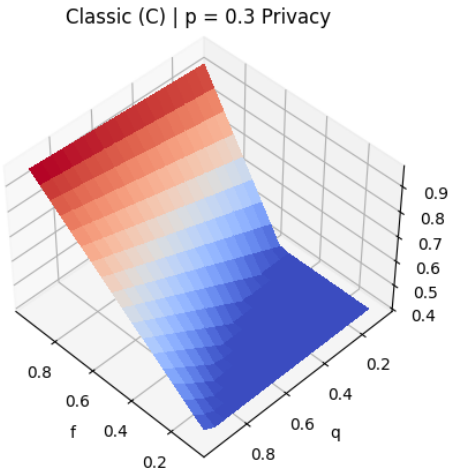
Insecurity plots

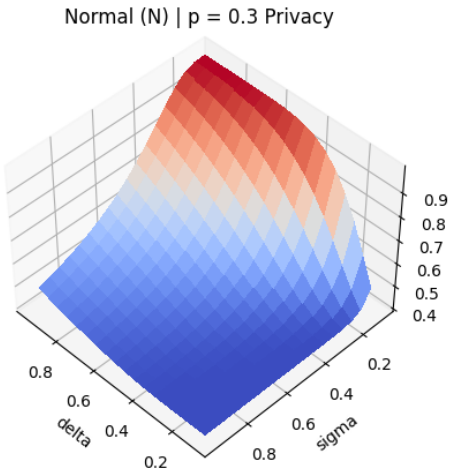


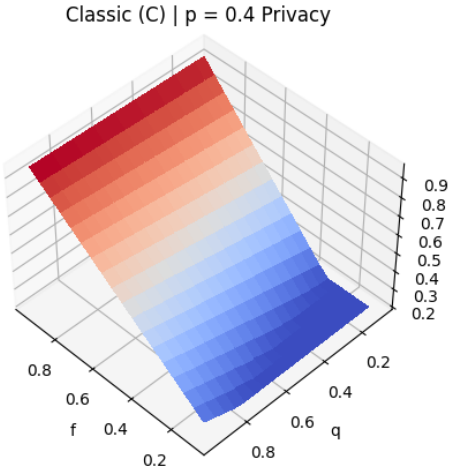


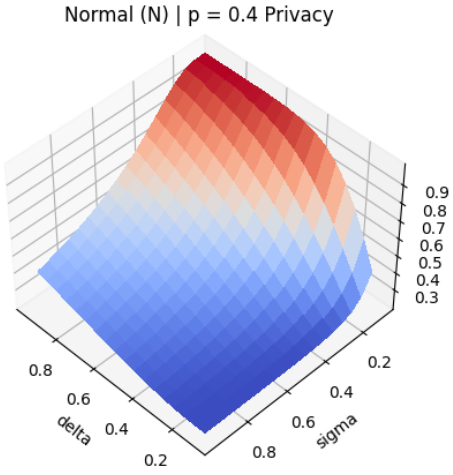


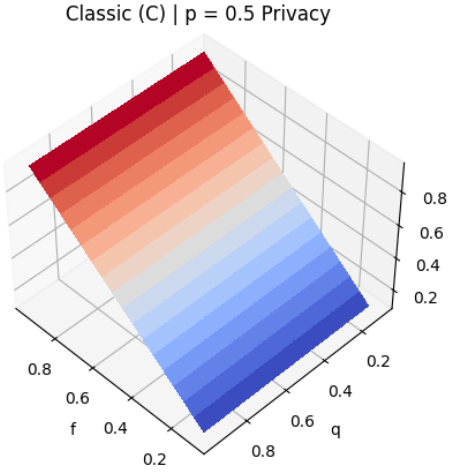


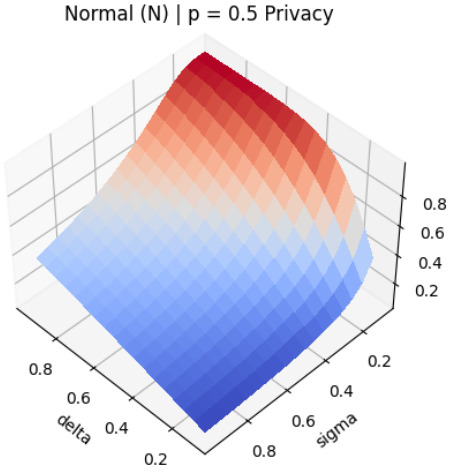

Classic (C) | p = 0.3 Privacy
0.9
0.8
0.7
0.6
0.5
0.4
0.8
0.6
f
0.4
0.2
0.2
0.4
0.6
q
0.8

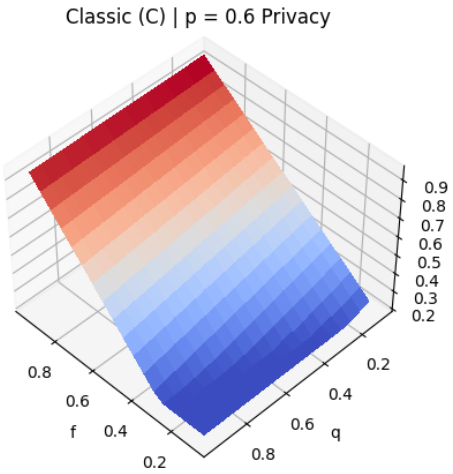

Normal (N) | p = 0.3 Privacy
0.9
0.8
0.7
0.6
0.5
0.4
0.8
0.6
delta
0.4
0.2
0.2
0.4
0.6
sigma
0.8

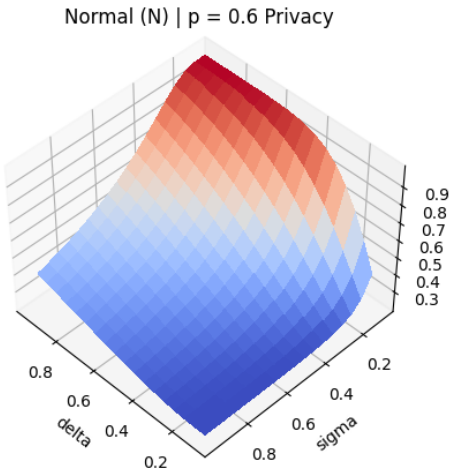

Classic (C) | p = 0.4 Privacy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.8
0.6
f
0.4
0.2
0.2
0.4
0.6
q
0.8

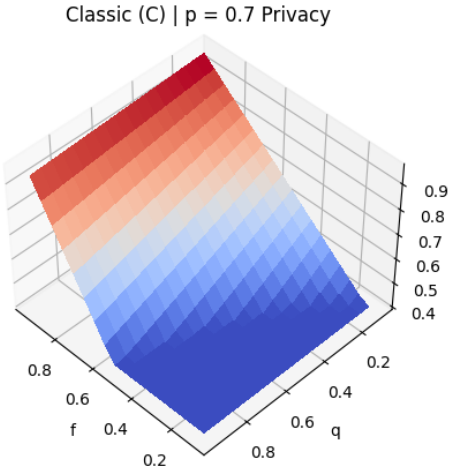

Normal (N) | p = 0.4 Privacy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.8
0.6
delta
0.4
0.2
0.2
0.4
0.6
sigma
0.8

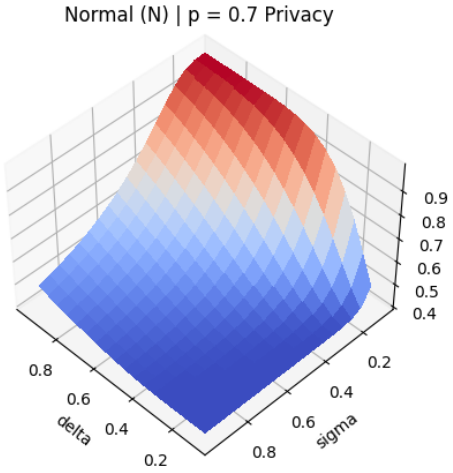

Classic (C) | p = 0.5 Privacy
0.8
0.6
0.4
0.2
0.8
0.6
f
0.4
0.2
0.2
0.4
0.6
q
0.8

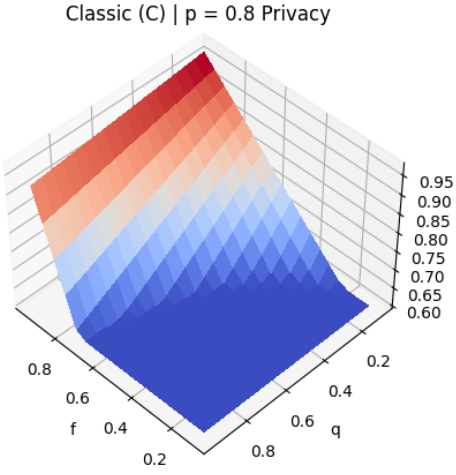

Normal (N) | p = 0.5 Privacy
0.8
0.6
0.4
0.2
0.8
0.6
delta
0.4
0.2
0.2
0.4
0.6
sigma
0.8

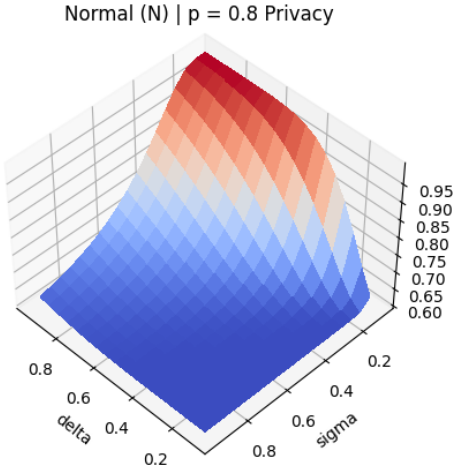

Classic (C) | p = 0.6 Privacy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.8
0.6
f
0.4
0.2
0.2
0.4
0.6
q
0.8

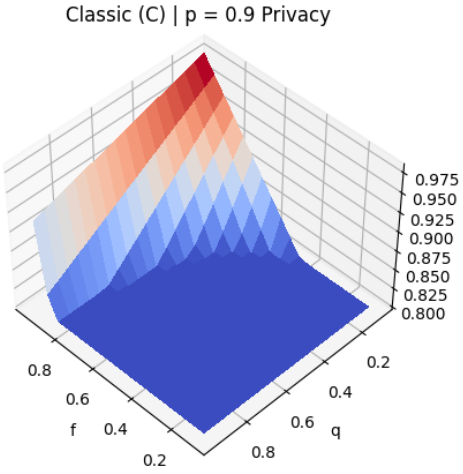

Normal (N) | p = 0.6 Privacy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.8
0.6
delta
0.4
0.2
0.2
0.4
0.6
sigma
0.8

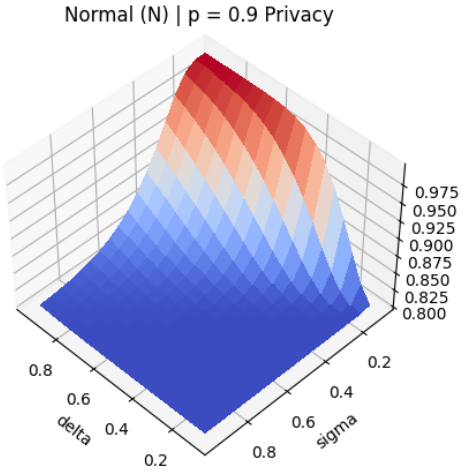

Classic (C) | p = 0.7 Privacy
0.9
0.8
0.7
0.6
0.5
0.4
0.8
0.6
f
0.4
0.2
0.2
0.4
0.6
q
0.8


Normal (N) | p = 0.7 Privacy
0.9
0.8
0.7
0.6
0.5
0.4
0.8
0.6
delta
0.4
0.2
0.2
0.4
0.6
sigma
0.8

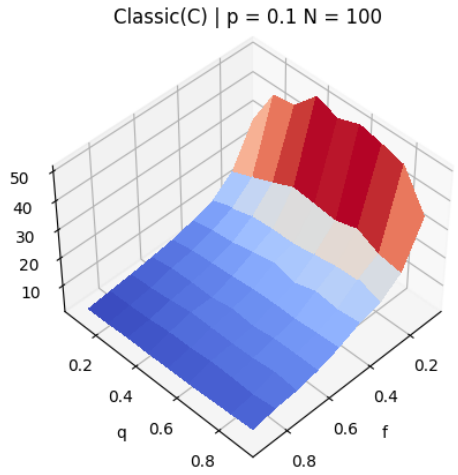

Classic (C) | p = 0.8 Privacy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.8
0.6
f
0.4
0.2
0.2
0.4
0.6
q
0.8

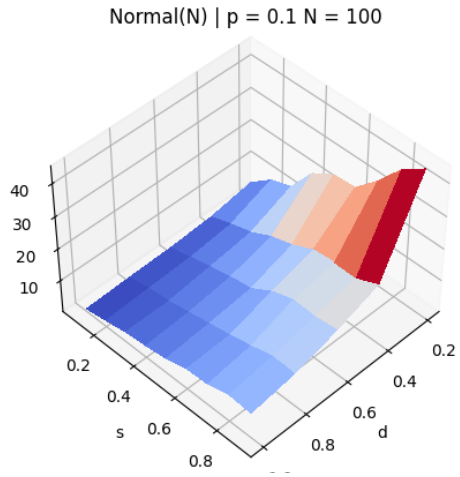

Normal (N) | p = 0.8 Privacy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.8
0.6
delta
0.4
0.2
0.2
0.4
0.6
sigma
0.8

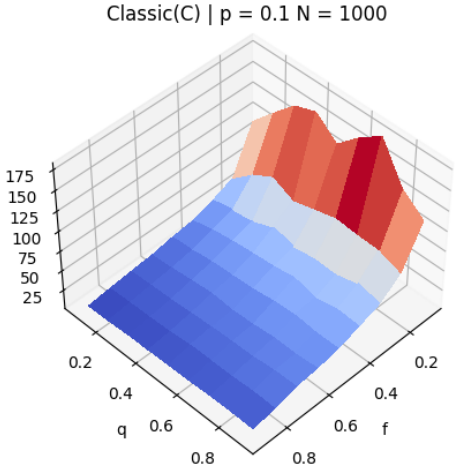

Classic (C) | p = 0.9 Privacy
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.8
0.6
f
0.4
0.2
0.2
0.4
0.6
q
0.8

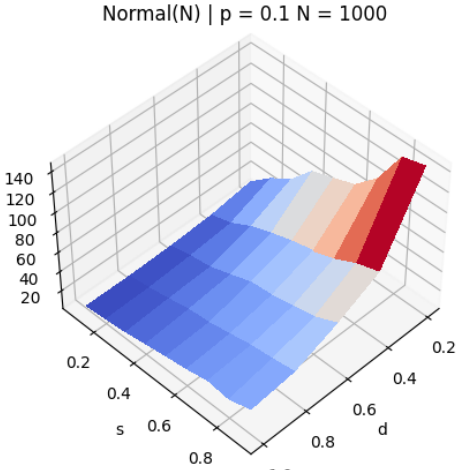

Normal (N) | p = 0.9 Privacy
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.8
0.6
delta
0.4
0.2
0.2
0.4
0.6
sigma
0.8

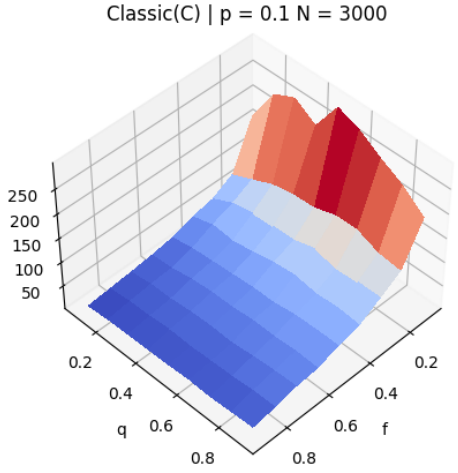
Accuracy plots

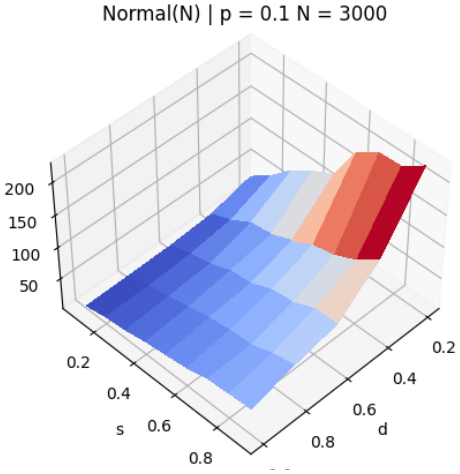


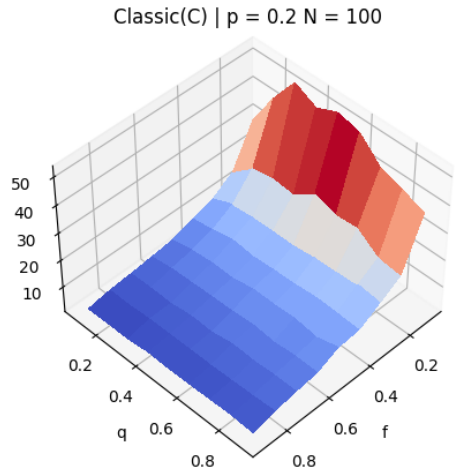


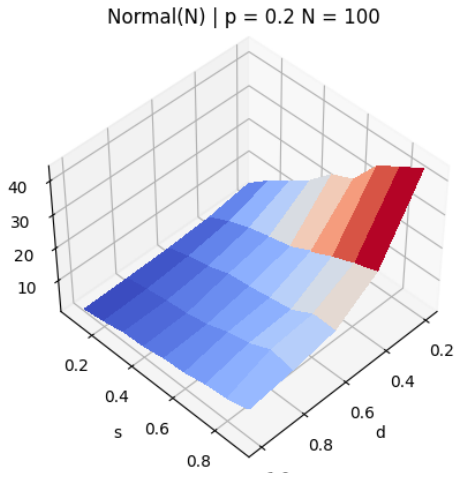


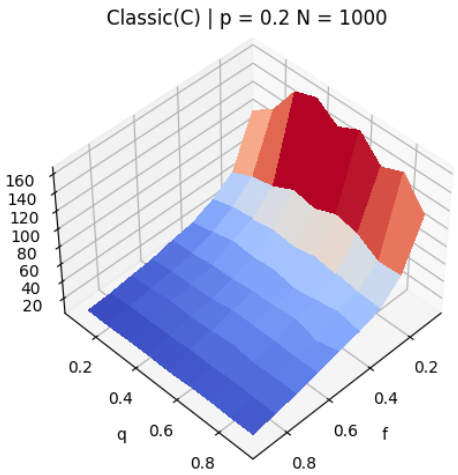


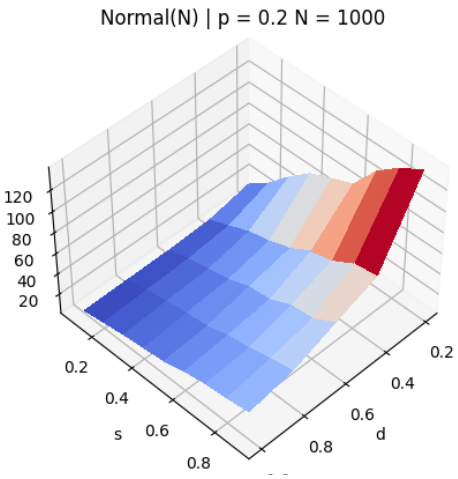


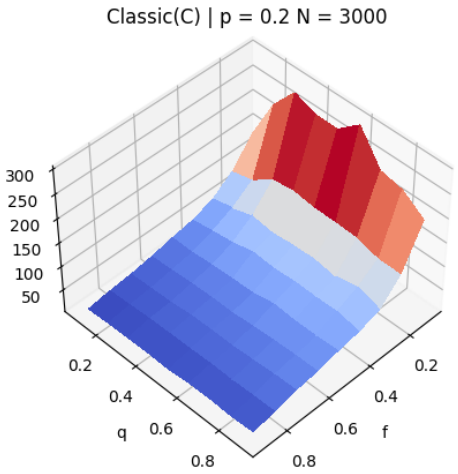


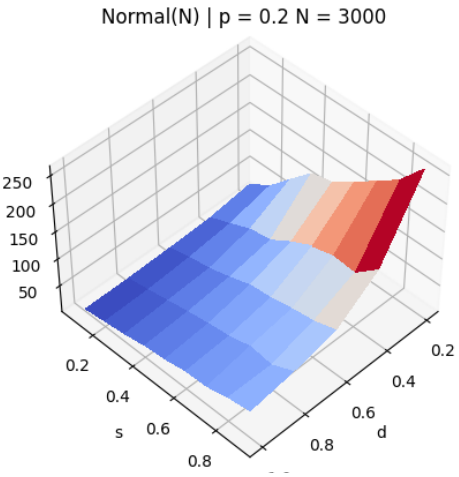

Classic(C) | p = 0.2 N = 100
50
40
30
20
10
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

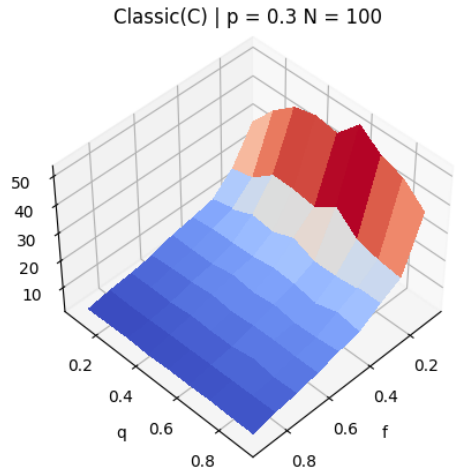

Normal(N) | p = 0.2 N = 100
40
30
20
10
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

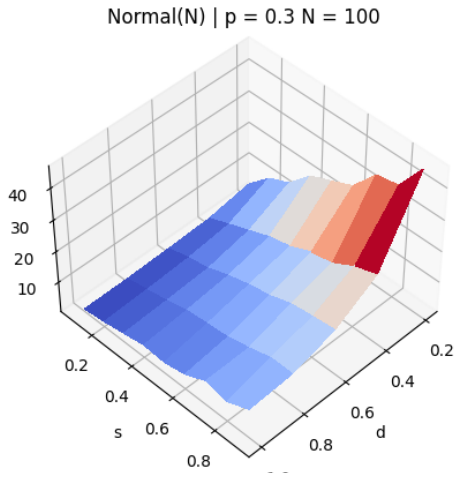

Classic(C) | p = 0.2 N = 1000
160
140
120
100
80
60
40
20
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

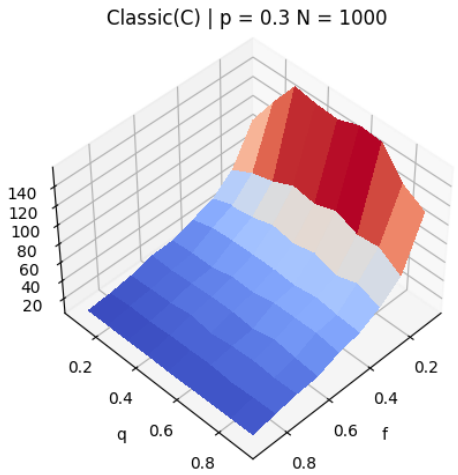

Normal(N) | p = 0.2 N = 1000
120
100
80
60
40
20
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

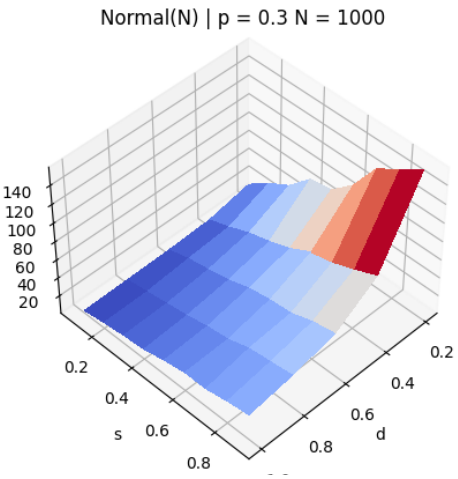

Classic(C) | p = 0.2 N = 3000
300
250
200
150
100
50
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

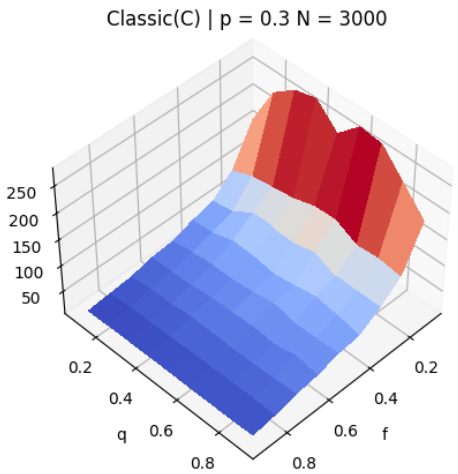

Normal(N) | p = 0.2 N = 3000
250
200
150
100
50
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

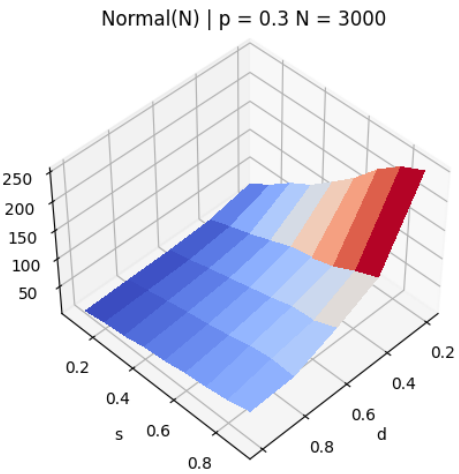

Classic(C) | p = 0.3 N = 100
50
40
30
20
10
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

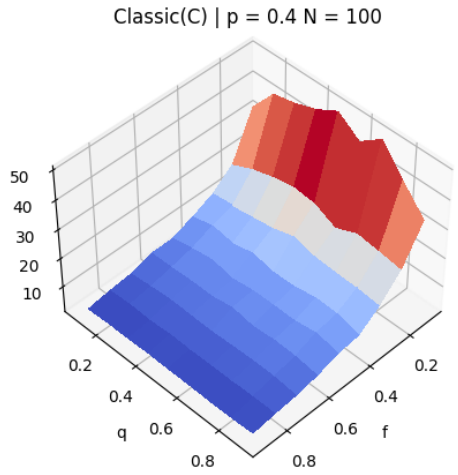

Normal(N) | p = 0.3 N = 100
40
30
20
10
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

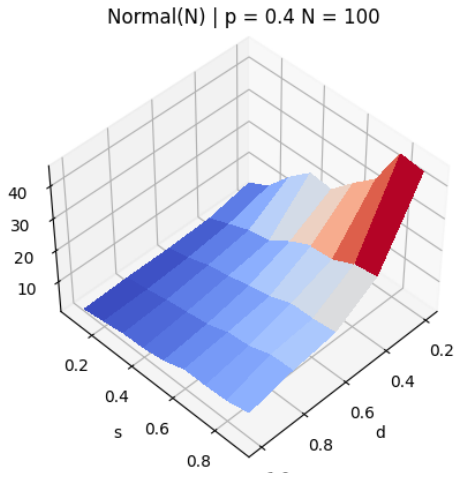

Classic(C) | p = 0.3 N = 1000
140
120
100
80
60
40
20
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

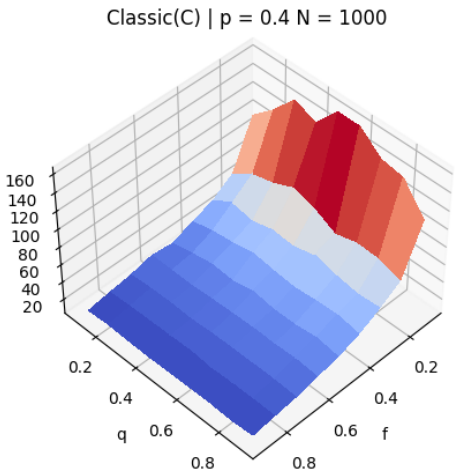

Normal(N) | p = 0.3 N = 1000
140
120
100
80
60
40
20
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

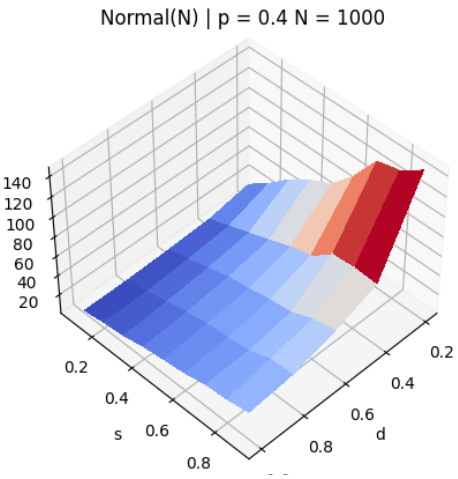

Classic(C) | p = 0.3 N = 3000
250
200
150
100
50
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

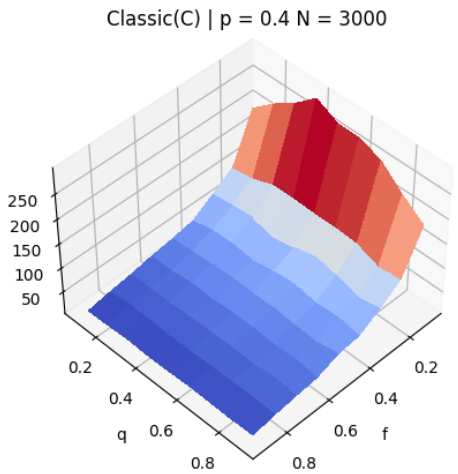

Normal(N) | p = 0.3 N = 3000
250
200
150
100
50
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

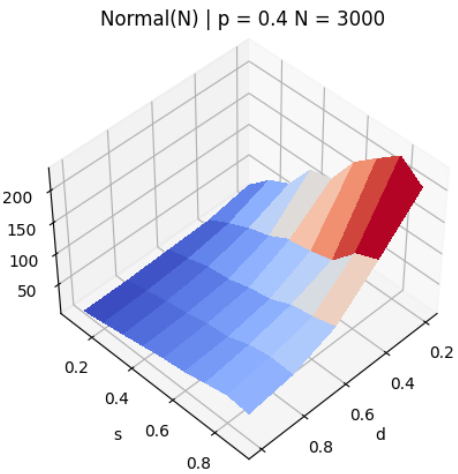

Classic(C) | p = 0.4 N = 100
50
40
30
20
10
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

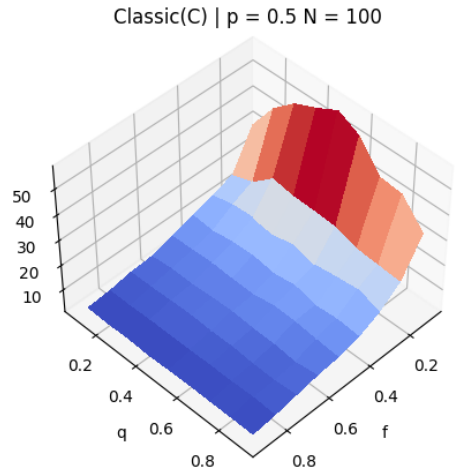

Normal(N) | p = 0.4 N = 100
40
30
20
10
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

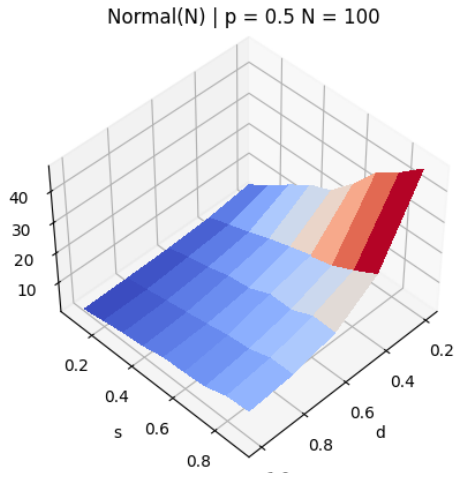

Classic(C) | p = 0.4 N = 1000
160
140
120
100
80
60
40
20
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

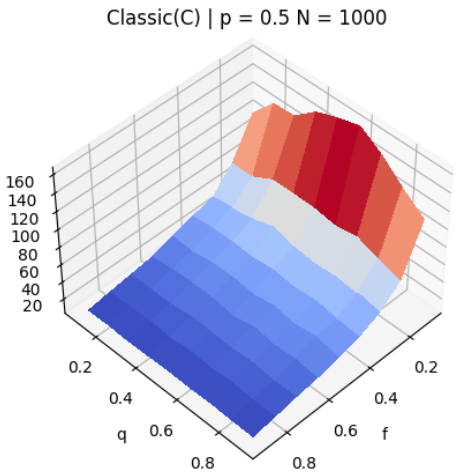

Normal(N) | p = 0.4 N = 1000
140
120
100
80
60
40
20
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

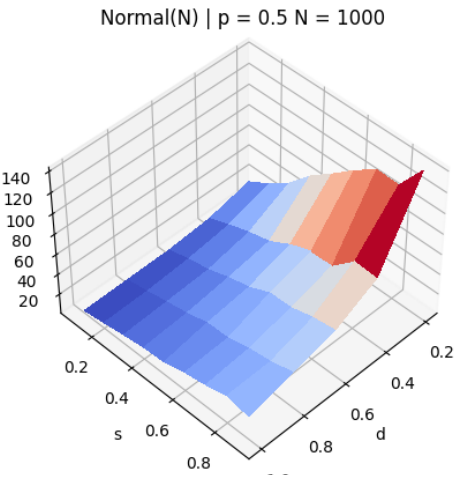

Classic(C) | p = 0.4 N = 3000
250
200
150
100
50
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

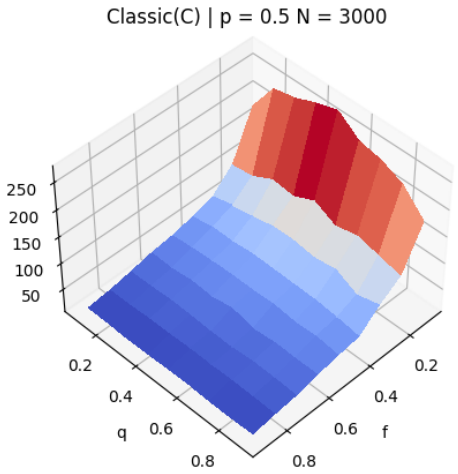

Normal(N) | p = 0.4 N = 3000
200
150
100
50
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

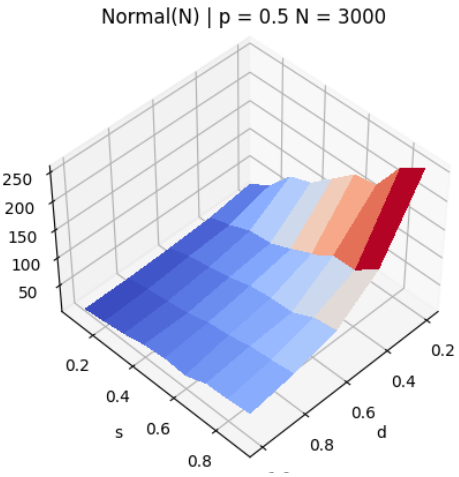

Classic(C) | p = 0.5 N = 100
50
40
30
20
10
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

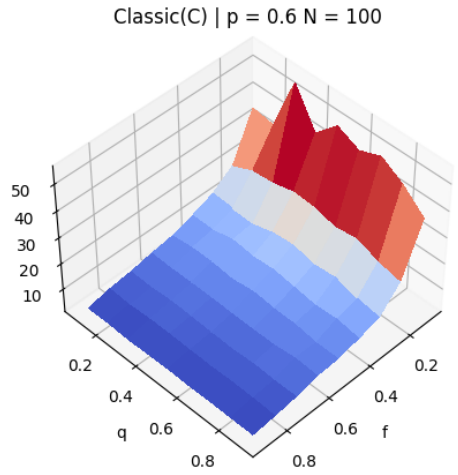

Normal(N) | p = 0.5 N = 100
40
30
20
10
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

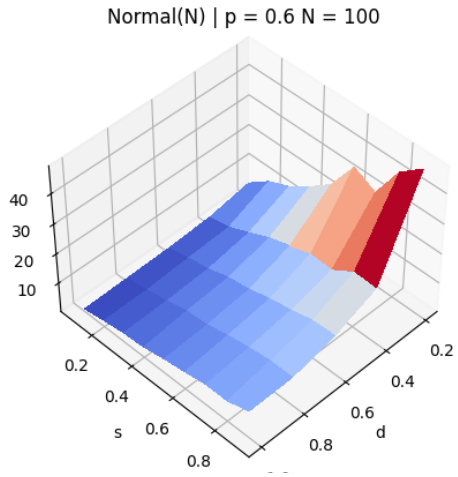

Classic(C) | p = 0.5 N = 1000
160
140
120
100
80
60
40
20
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

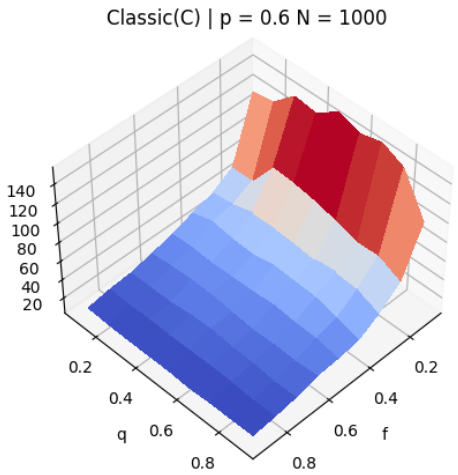

Normal(N) | p = 0.5 N = 1000
140
120
100
80
60
40
20
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

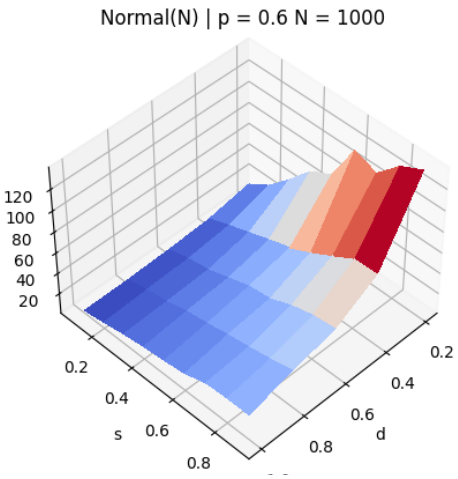

Classic(C) | p = 0.5 N = 3000
250
200
150
100
50
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

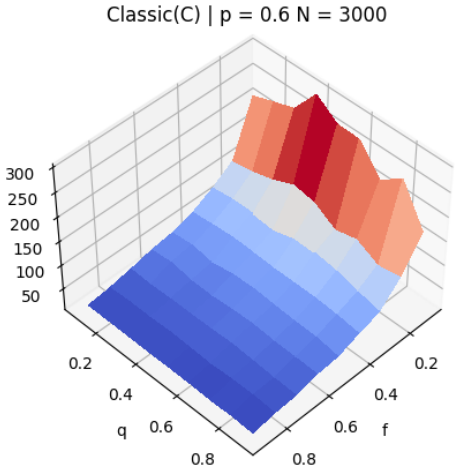

Normal(N) | p = 0.5 N = 3000
250
200
150
100
50
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

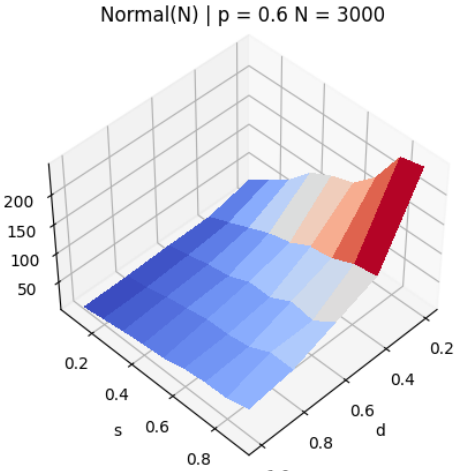

Classic(C) | p = 0.6 N = 100
50
40
30
20
10
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

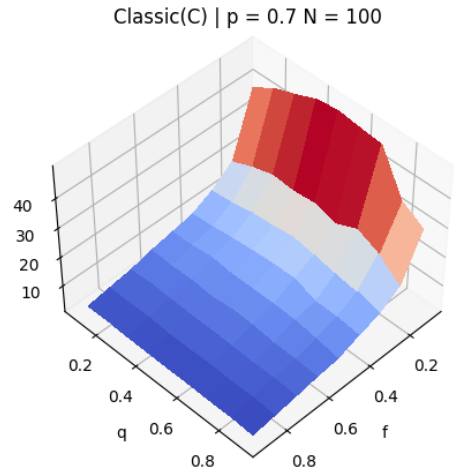

Normal(N) | p = 0.6 N = 100
40
30
20
10
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

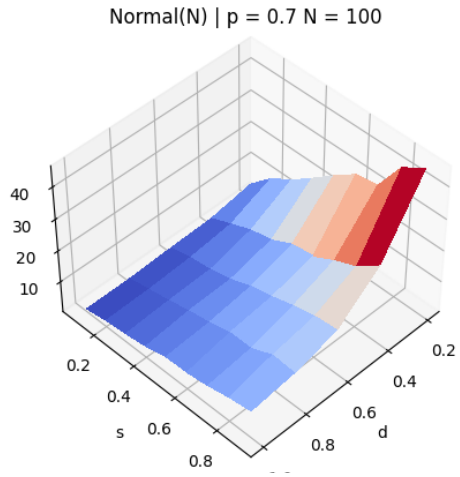

Classic(C) | p = 0.6 N = 1000
140
120
100
80
60
40
20
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

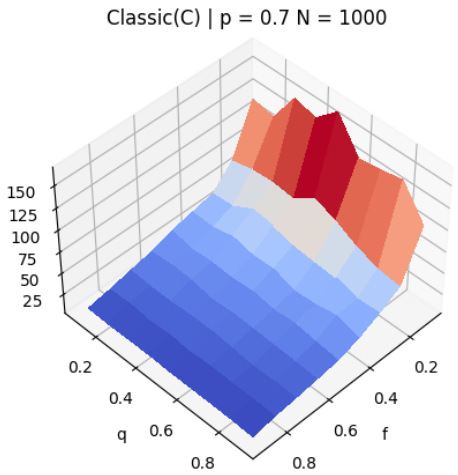

Normal(N) | p = 0.6 N = 1000
120
100
80
60
40
20
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

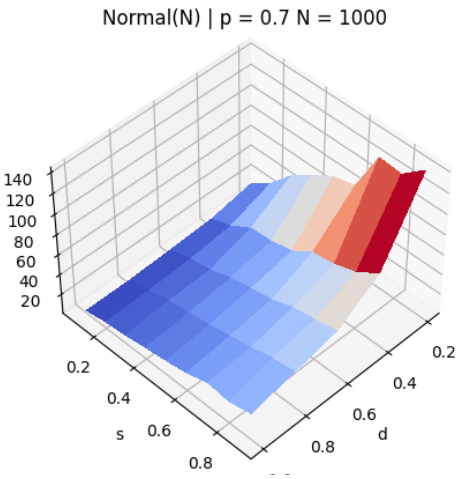

Classic(C) | p = 0.6 N = 3000
300
250
200
150
100
50
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

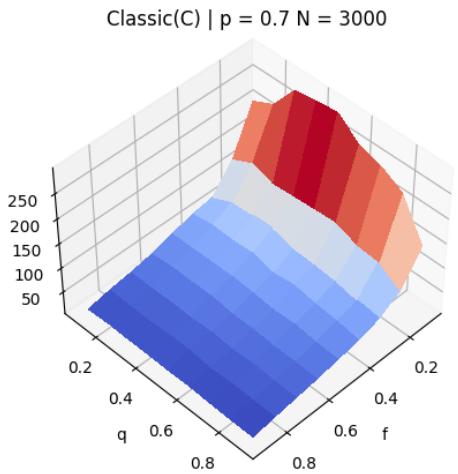

Normal(N) | p = 0.6 N = 3000
200
150
100
50
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

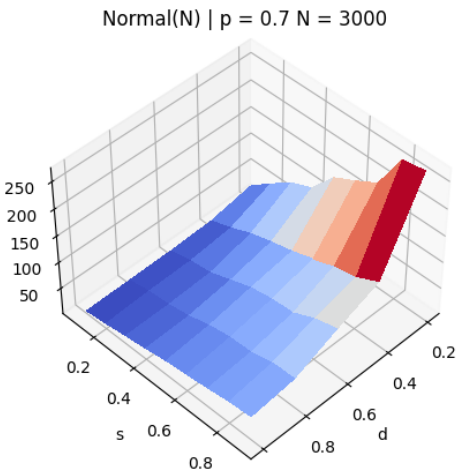

Classic(C) | p = 0.7 N = 100
40
30
20
10
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

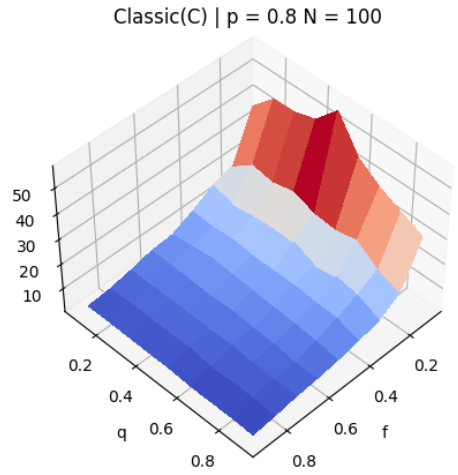

Normal(N) | p = 0.7 N = 100
40
30
20
10
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

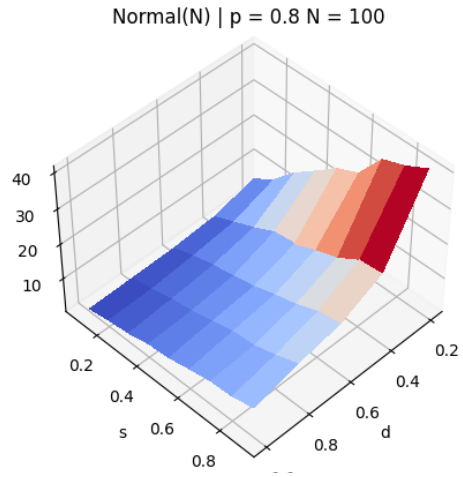

Classic(C) | p = 0.7 N = 1000
150
125
100
75
50
25
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

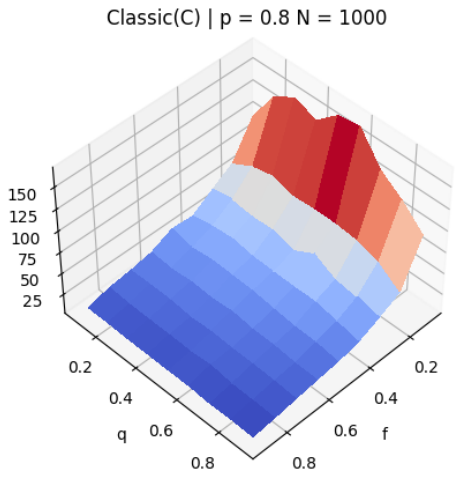

Normal(N) | p = 0.7 N = 1000
140
120
100
80
60
40
20
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

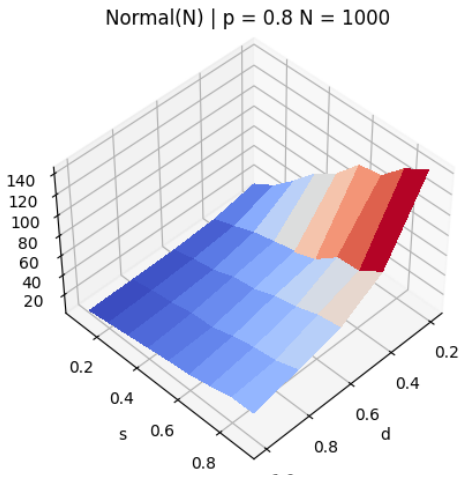

Classic(C) | p = 0.7 N = 3000
250
200
150
100
50
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

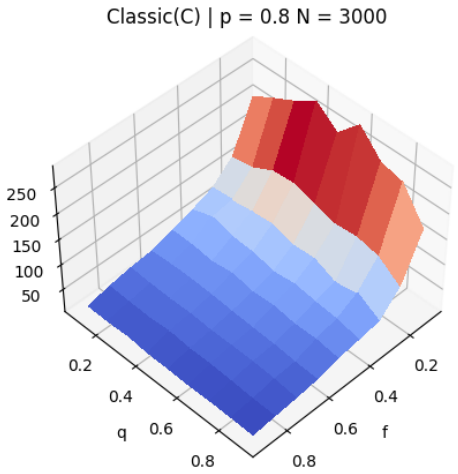

Normal(N) | p = 0.7 N = 3000
250
200
150
100
50
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

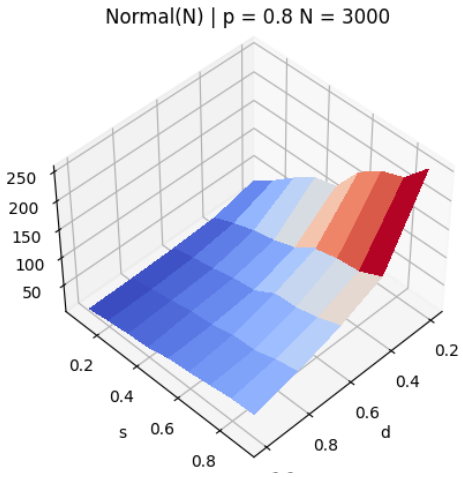

Classic(C) | p = 0.8 N = 100
50
40
30
20
10
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

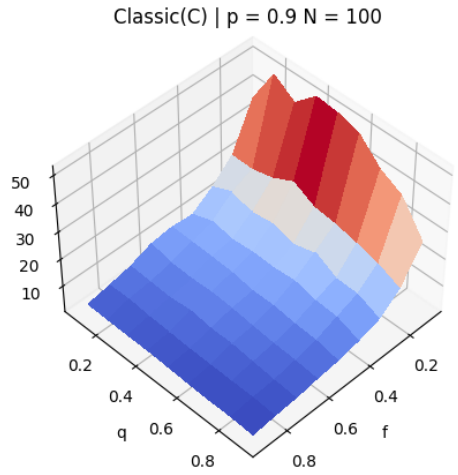

Normal(N) | p = 0.8 N = 100
40
30
20
10
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

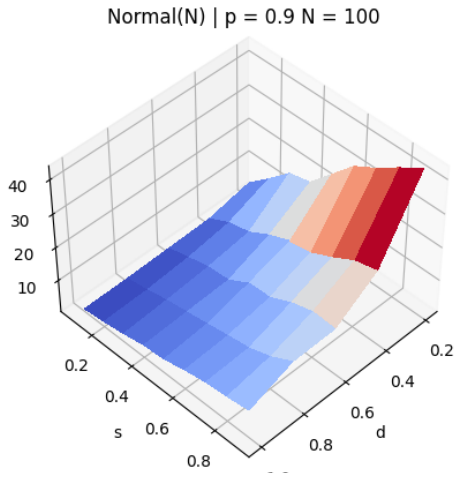

Classic(C) | p = 0.8 N = 1000
150
125
100
75
50
25
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

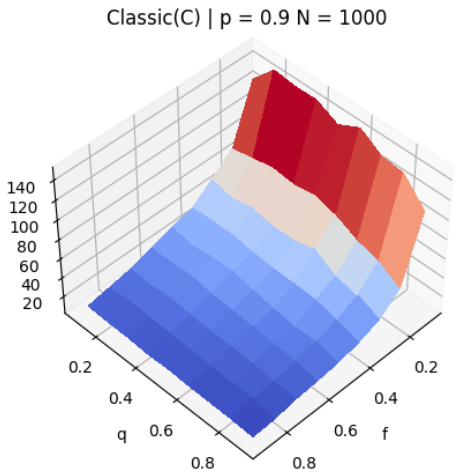

Normal(N) | p = 0.8 N = 1000
140
120
100
80
60
40
20
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

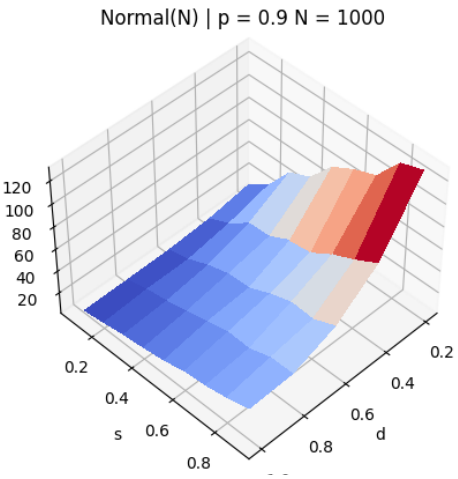

Classic(C) | p = 0.8 N = 3000
250
200
150
100
50
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8

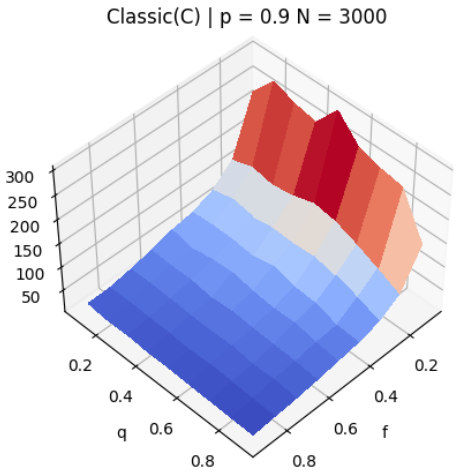

Normal(N) | p = 0.8 N = 3000
250
200
150
100
50
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8

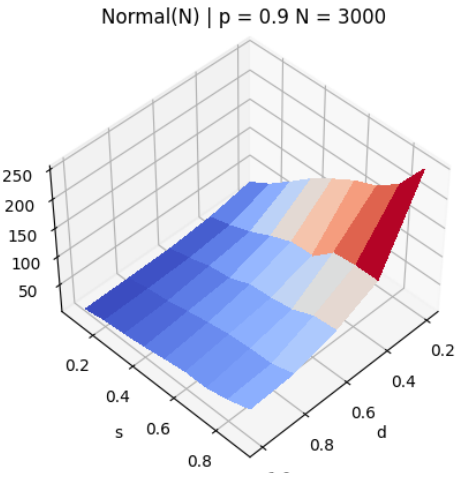

Classic(C) | p = 0.9 N = 100
50
40
30
20
10
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8


Normal(N) | p = 0.9 N = 100
40
30
20
10
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8


Classic(C) | p = 0.9 N = 1000
140
120
100
80
60
40
20
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8


Normal(N) | p = 0.9 N = 1000
120
100
80
60
40
20
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8


Classic(C) | p = 0.9 N = 3000
300
250
200
150
100
50
0.2
0.4
q
0.6
0.8
0.2
0.4
0.6
f
0.8


Normal(N) | p = 0.9 N = 3000
250
200
150
100
50
0.2
0.4
s
0.6
0.8
0.2
0.4
0.6
d
0.8