Universität Potsdam
Mathematisch-Naturwisssenschaftliche Fakultät
Institut für Physik und Astronomie

Computational Physics

Stochastic Resonance

Author: Alexander Putz
Matrikel-No: 763265
Author: Christian Gößl
Matrikel-No.:762627

corrector: Prof. Dr. Arkady Pikovsky

July 19, 2019

Contents

1 Introduction

When taking a look at phenomena like weather, biological systems, etc., one does always have to take unknown, not determinable quantities into account. This is usually addressed by adding a statistical part into the equations, which seem to represent the real behavior very well. We want to investigate a statistical system, that describes a random moving particle with only two states. The hopping rate between these two states is described by the so called Kramers formula. Additionally a periodic force and a white Gaussian noise acts on the particle. This leads to a stochastic differential equation. By carefully tuning noise and driving force, one can observe the phenomena of stochastical resonance. Applications can be nothing less than our climate with periods of warm and cold states.

2 Stochastic Resonance Equation

In our generic model we investigate the time evolution of a stochastic variable x. In our work we want to simulate a particle, that is performing a random walk in a fluid, namely the well known brownian motion. Here we investigate only the drift in x direction. Additionally the particle is moving in a potential $V(x)$ and we can switch on a driving force. It has two minima, where the particle can be located. These two states are separated trough a potential wall ΔV. The time evolution is described as follows:

$$x_t = -V_x(x) + A\cos(\omega t + \phi) + \sigma\xi(t) \tag{1}$$

The subscript variable means a derivative $x_t = \mathrm{d}x/\mathrm{d}t$. As usual: t time variable, A amplitude and ω frequency of the oscillator and $\xi(t)$ as noise, it is the temporal derivative of a Wiener Process.

$$\xi(t) = \frac{\mathrm{d}W}{\mathrm{d}t} \tag{2}$$

The potential function $V(x)$ of the stochastic variable x:

$$V(x) = -\frac{1}{2}x^2 + \frac{1}{4}x^4$$

We use an oscillator to get a triggered resonance. In our model we work with an additional Gaussian white noise and with the auto-correlation we get

$$\langle\xi(t)\xi(0)\rangle = 2D\delta(t) \tag{3}$$

with σ the noise factor, hence the variance of noise is $\sqrt{2D} = \sigma$. In our investigation we set the initial phase to

$$\phi = 0\,.$$

We add all relations together and get for our simulation the main equation:

$$x_t = x^3 - x + A\cos(\omega t) + \sqrt{2D}\xi(t) \tag{4}$$

The two minima of the potential function are $x_\pm = \pm 1$ and hence we have $\Delta V = 1/4$.

2.1 Numerical solution method: Euler-Maruyama

In general, the stochastic differential equation can be expressed in the differential form:

$$\mathrm{d}x = a(t,x)\mathrm{d}t + b(t,x)\mathrm{d}W \tag{5}$$

The initial value problem is calculated through

$$x_{i+1} = x_i + a(t_i,x_i)\Delta t_i + b(t_i,x_i)\Delta W_i \tag{6}$$

with $a(t_i,x_i)$ as function for the non-stochastic part

$$a(t_i,x_i) = x_i - x_i^3 + A\cos(\omega t_i)$$

and $b(t_i,x_i)$ as function for the stochastic part

$$b(t_i,x_i) = \sqrt{2D}\,.$$

For ΔW_i we need a random distributed variable z_i. We are using here the normal distribution $F(\mu,q)$, which is white noise, with

$$\begin{aligned} z_i &\in F(0,1) \\ \Delta W_i &= z_i\sqrt{\Delta t_i} \end{aligned}$$

Summing up all equations, we receive:

$$x_{i+1} = x_i + (x_i - x_i^3 + A\cos(\Omega t_i))\Delta t_i + z_i\sqrt{2D\Delta t_i} \tag{7}$$

2.2 Kramers formula - Kramers rate

Kramers formula is used to give a time scale for the noise induced hopping. For the switching rate r_K we have the equation

$$r_\mathrm{K}(D) = \frac{1}{\sqrt{2}\pi}\exp\left(-\frac{\Delta V}{D}\right) . \tag{8}$$

With some rearrangements and $r_\mathrm{K} = 1/T_\mathrm{K}$ we get

$$\ln(T_\mathrm{K})(D) = \Delta V\frac{1}{D} + \ln(\sqrt{2}\pi) \tag{9}$$

Further we insert the preset values and the equation that we want to prove

$$\ln(T_\mathrm{K})(D) \approx 0.25\left(\frac{1}{D}\right) + 1.49130 . \tag{10}$$

2.3 Stochastic resonance

The main aspect of our work is the stochastic resonance. The resonance comes into account, when we turn on the driving force. It's evolving in time and the hopping between the states can be described by the Kramer formula. Naturally we get a resonance, when the frequency of Kramer and the driving force are in resonance. Later on, we will see, that our first impression is not entirely true. The particle is moving randomly, but with the periodic force we want to investigate how the random process is depending on the frequency ω and noise strength D.

For that we study the phase and amplitude of the time evolution. So we want to know the response of the system, hence we need a Fourierstransformation - FFT, to find the phase and amplitude of the signal at frequency of the driving force ω. We get from the FFT the Fouriercoefficents a_k, b_k and can compute the amplitude A and phase ϕ.

$$A = \sqrt{a_k^2 + b_k^2} \quad \phi = \arctan\left(\frac{a_k}{b_k}\right)$$

In the paper[1], we find the equation for the stochastic resonance.

$$\bar{\phi}(D) = \arctan\left(\frac{\omega}{2r_\mathrm{K}(D)}\right) \tag{11}$$

The Maximum resonance $\bar{\phi}_\mathrm{max}$ is reached at $\omega = \pi r_\mathrm{K}$.

$$\bar{\phi}_\mathrm{max}(\omega = \pi r_\mathrm{K}) = \arctan\left(\frac{\pi}{2}\right) \tag{12}$$

3 Numerical investigation of our stochastic DGL

3.1 Setup the simulation

The data is generated via numerical solution(see chapter 2.1) of the stochastical differential equation (4). This is done via the `set_Data()` function. In this function the data are calculated for a given set of parameters: frequency of the driving force ω, amplitude of the driving force A, noise intensity D, time step Δt, amount of iteration N, which were transferred during function `call()`. Later we decided to calculate time step with

$$\Delta t = \frac{2\pi L}{N\omega}$$

Because for the computation of nonlinear dependency of the amplitude A, we set the duration of simulation to $T \cdot L$, where T describes the period of the driving force and the factor L gives us a time evolution until the period ended. In our investigations we often use a mean of a physical quantity, due to get better stochastic relevant results.

At first `init()` function is initializing streams for output. The first call of `output()` sets the boundary conditions at $t_0 = 0$ and $x(t_0) = 0$. After that, `N` iterations of the SDE are done and stored in the vector `x`. The received data can now be used for further investigations. In graphic 1, an exemplary set of data is shown.

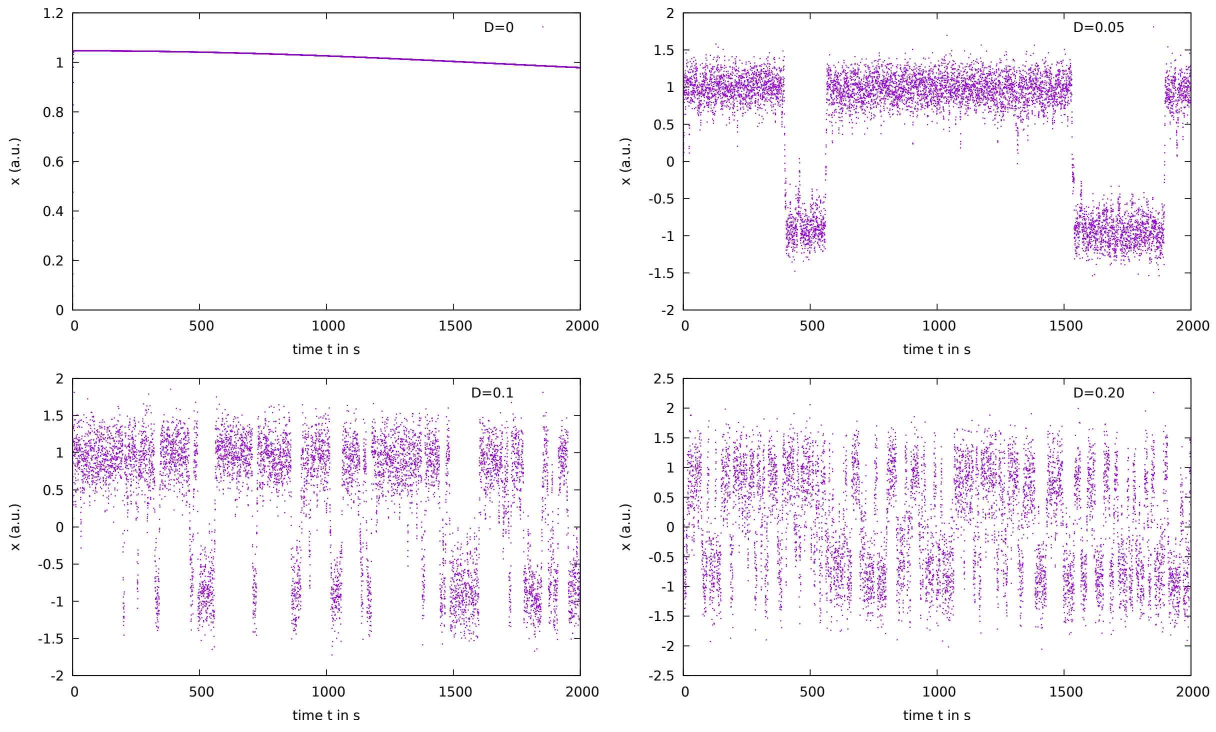


Figure 1: The data was gathered for an amplitude $A = 0.1$ and a frequency $\omega = 0.001$. The graphs, display data for different noise ranging from 0 to 0.25. Between $D = 0{,}05$ and $D = 0.1$, one can see a oscillation of the response signal. For $D = 0$, there is hardly an oscillation visible

3.2 Verification of the Kramers formula

Here we want to prove the Kramers formula. The Kramers formula describes the switching rate r_K between the two potential states. In our simulation we have the states up and down. From the amount of states n_K we can calculate the period T_K. We calculated the mean value of the $\ln \bar{T}_K$ for specific times M

$$\ln \bar{T}_K = \frac{N \Delta t M}{n_K} \,.$$

So we need an algorithm, that can detect these states of our simulation. The program detects and counts the changings with our predefined rules in the function `kramer()`. The principle goes like this: Whenever x_i is in up or down state, the function checks, whether there was a transition from the opposite state or x_i remains in the state it was before. After more calculations the values goes up and we have a up state. Then the procedure starts again as before. We indicate the states with the Boolean variable `mess` and when the values either for up $x_i > 0.99$ or down $x_i < -0.99$. At the beginning it is set to `false`. It changes to `true`, when the next down state is reached. Our convention for the states is up with `mess = false` and down with `mess = true`. You can see the whole code in the appendix. For the verification of Kramer formula we use the equation (10). It is fitted to our calculated values $\ln(T_K)(D)$.

$$\ln(T_f)(D) = a\left(\frac{1}{D}\right) + b$$

$$L = 30\,, N = 10^6\,, A = 0\,, \omega = 0.5\,, D_{\text{init}} = 0.1\,, D_{\text{end}} = 0.5\,, M = 20$$

See the figure 2 for the results.

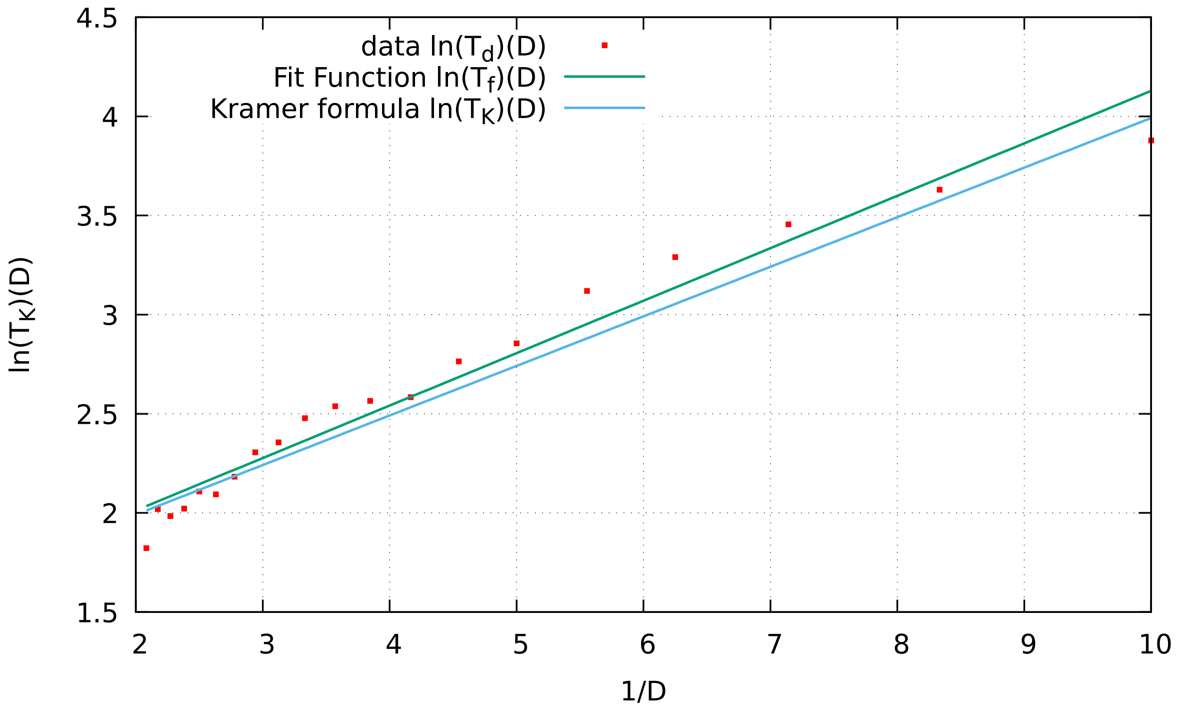


Figure 2: Comparison between the fitting function $\ln(T_f)(D)$ and the calculated function $\ln(T_K)(D)$. red squares: the calculated values, green line: fit function, blue line: Kramer formula. The slope of the fit function has a good agreement with the Kramer formula. It is slightly shifted upwards.

For our fit function we get the following parameters:

$$\ln(T_f)(D) \approx 0.26454\left(\frac{1}{D}\right) + 1.48301 \qquad \ln(T_K)(D) \approx 0.25\left(\frac{1}{D}\right) + 1.49130$$

Our result gives us a good agreement with the predicted model of the Kramer formula.

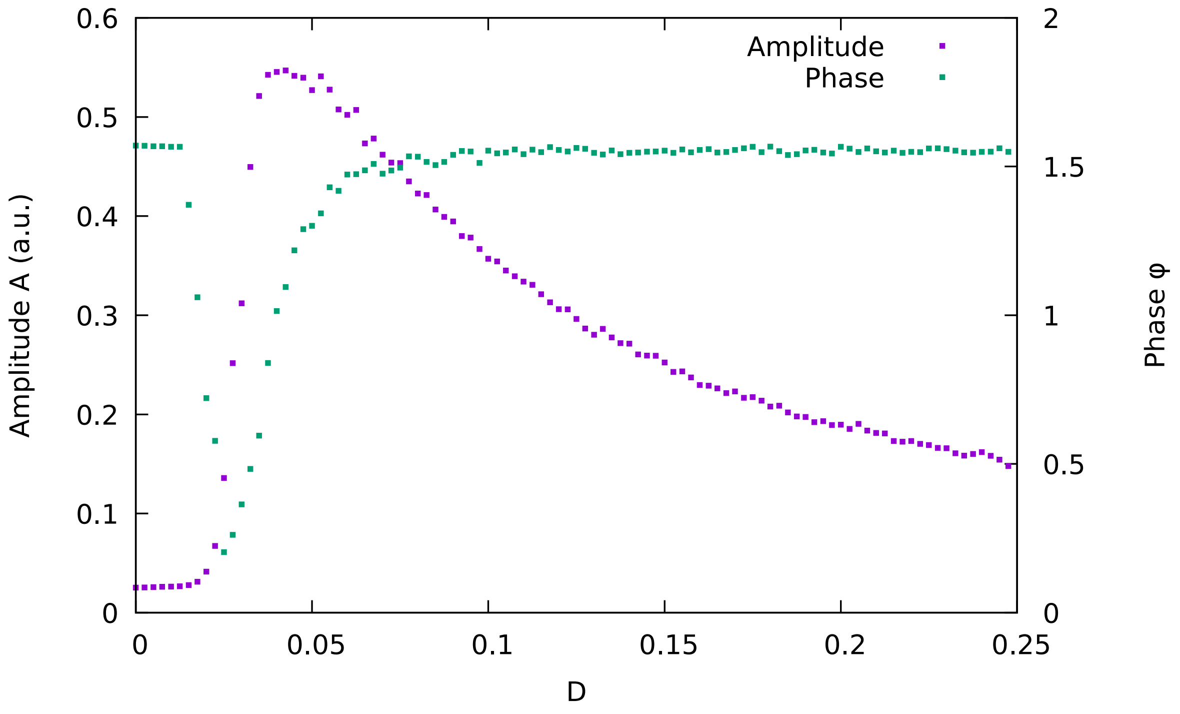


Figure 3: amplitude-phase plotted against noise intensity D for $A = 0.1$ and $\omega = 0.001$

3.3 Response examination in dependence of noise

The beforehand gathered data (see section 3.1) are Fourier-Transformed at the periodic driving frequency ω. The Fourier-Transformation is done via the function `FFT()`.Varying the noise intensity D and determining the output amplitude of the Fourier components, one gets in figure 3 the shown trend. As one can see, there is a maximum amplitude at a noise of about $D = 0.04$. This is called "stochastical resonance". At this noise strength, the Kramers rate r_K and the periodic force are in resonance. The amplitude even gets stronger than without noise. If the noise gets even stronger, the switching is driven by the noise and the Kramers rate gets smaller. Therefore the amplitude at the periodic forcing ω gets smaller. Additionally one can observe a drop in phase shift of the sin and cos components. Out of resonance the phase shift is stable at $\pi/2$, but in resonance the phase shift gets reduced to about 0.2. Taking a closer look at the Fourier coefficients a_k and b_k in figure 5 unveils the phase shifts shape. The cos component b_k shows a peak at the rising flank of the amplitude. This is due to the input signal, which is also a cos function. In contrast to that, the sin component follows the amplitudes shape. As already explained in chapter 2.2, the matching condition(resonant case) is $\omega = \pi r_K$. Inserting the condition in equation (11) yields

$$\bar{\phi}_{\max} = \arctan\left(\frac{\pi}{2}\right) = 1 \tag{13}$$

This means, there is a fixed phase shift in the resonant case, which can be confirmed with figure 7. The maximum response amplitude is always at a phase shift of 1 (Caution, do not mix with the phase shifts minimum).

To get statistical data, this analysis was performed 10 times using the function `Fourier_task()` and averaged. The function does also return the amplitudes and phase shifts variance. This can be seen in figure 4 . Before reaching the resonant state, the variance for both amplitude and phase shift is nearly not existent compared to the variance at higher noise rates. This can be ascribed to the fact, that the oscillation is a statistical process and the variance will get smaller with increasing iteration numbers.

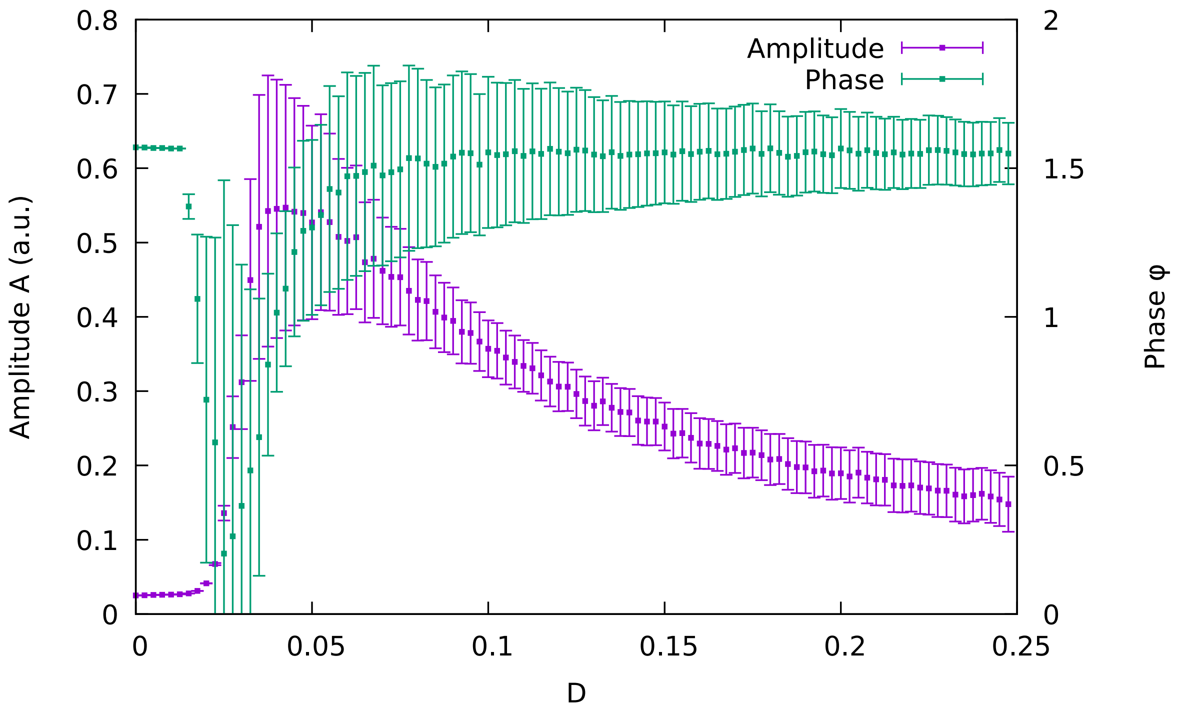


Figure 4: amplitude-phase plotted against noise intensity D for $A = 0.1$ and $\omega = 0.001$ with errorbars (variance). The number of iterations was 10

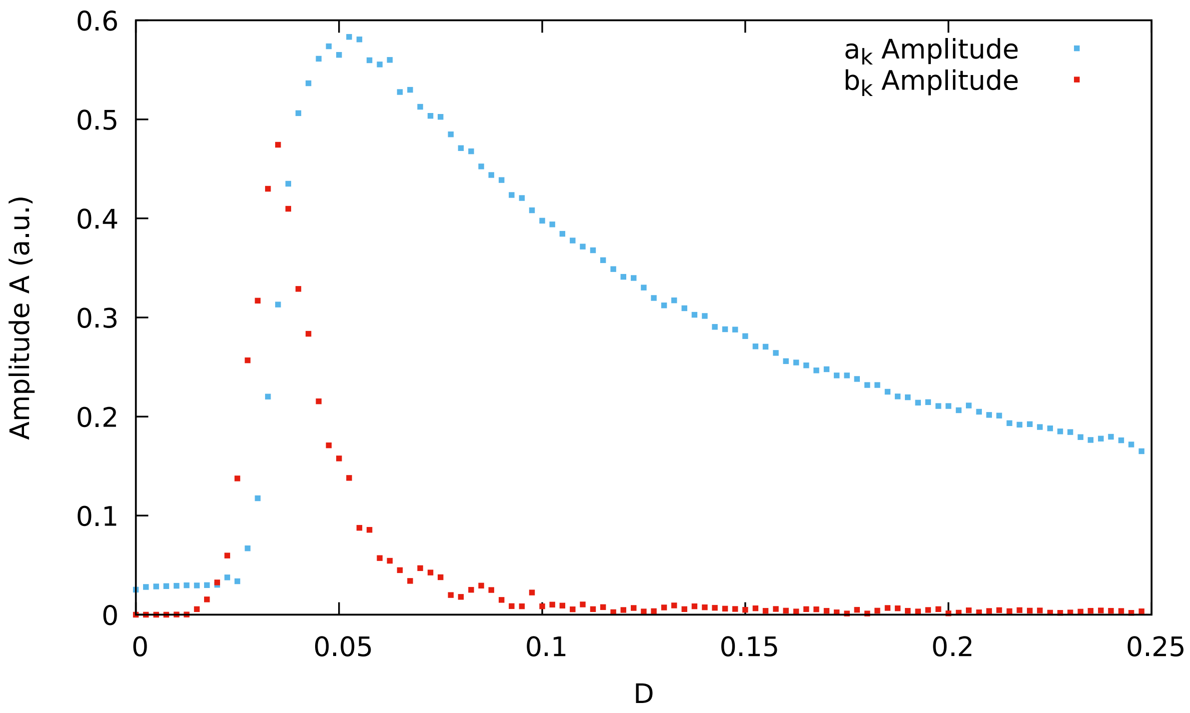
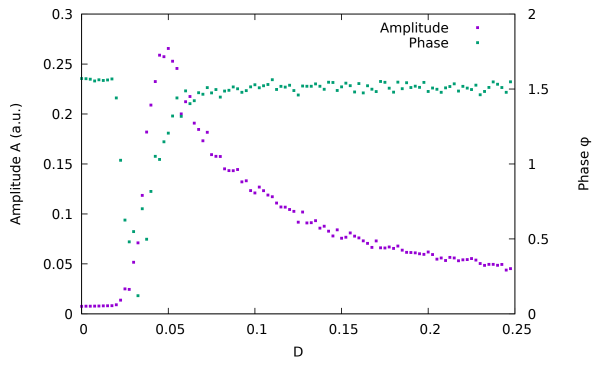


Figure 5: coefficients a_k and b_k in dependency of noise intensity D for $A = 0.1$ and $\omega = 0.001$

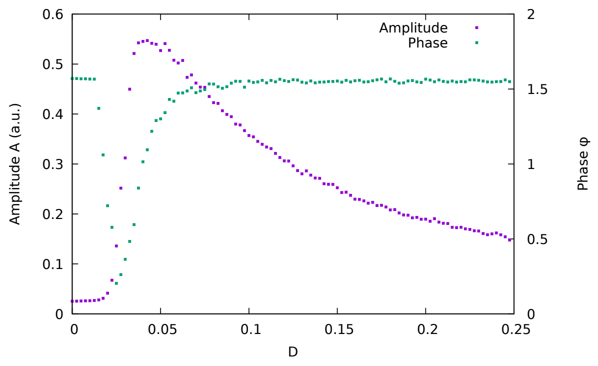
3.4 Examine non-linear properties of the signals amplitude A

In this part we want to inspect the non-linear properties of the signal amplitude A_{out}. For this task the algorithms and functions from chapter 3.3 can be taken. The input signals amplitude A_{in} was varied from 0.03 to 2.0 as can be seen in figure 7. Also for comparison all plots can be seen in figure 8. The systems amplitude rises with increasing signal amplitude. The increase is not linear. Increasing the signals amplitude from $A = 0.03$ to $A = 0.1$ (factor of about 3), increases the systems amplitude by a factor of about 2. A further increase of the signals amplitude from 0.1 to $A = 1$ (factor of 10), results in a much smaller increase of about 45%. Also, after reaching a certain level, the resonance peak shifts to lower noise intensities with rising signal amplitude. It seems like the peak shifts to "‘negative"’ noise, which would be nonphysical.

A possible explanation could be, that with a high signal strength the system is not anymore governed by the noise and therefore stochastical resonance occurs no longer. Also, the role of potential diminishes. In the presented case this is somewhere between $A = 0.2$ and $A = 0.4$.



(a) signal amplitude $A = 0.03$



(b) signal amplitude $A = 0.1$

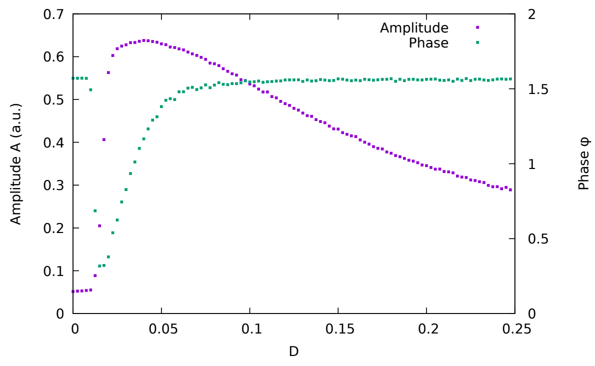
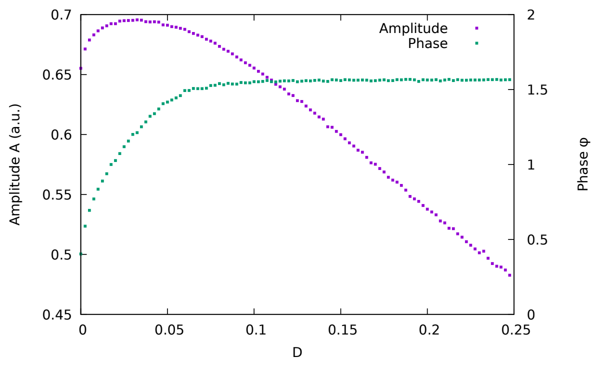
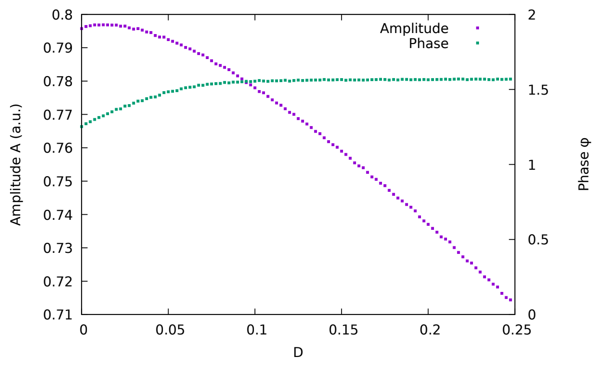


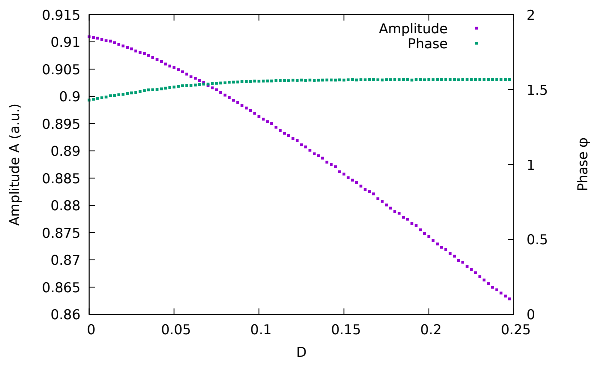
Figure 6: signal amplitude $A = 0.2$



(a) signal amplitude $A = 0.4$



(b) signal amplitude $A = 1.0$



(c) signal amplitude $A = 2.0$

Figure 7: Response amplitude for different signal amplitudes

We put the maxima of all plots together in one plot over the input amplitude see figure 9.

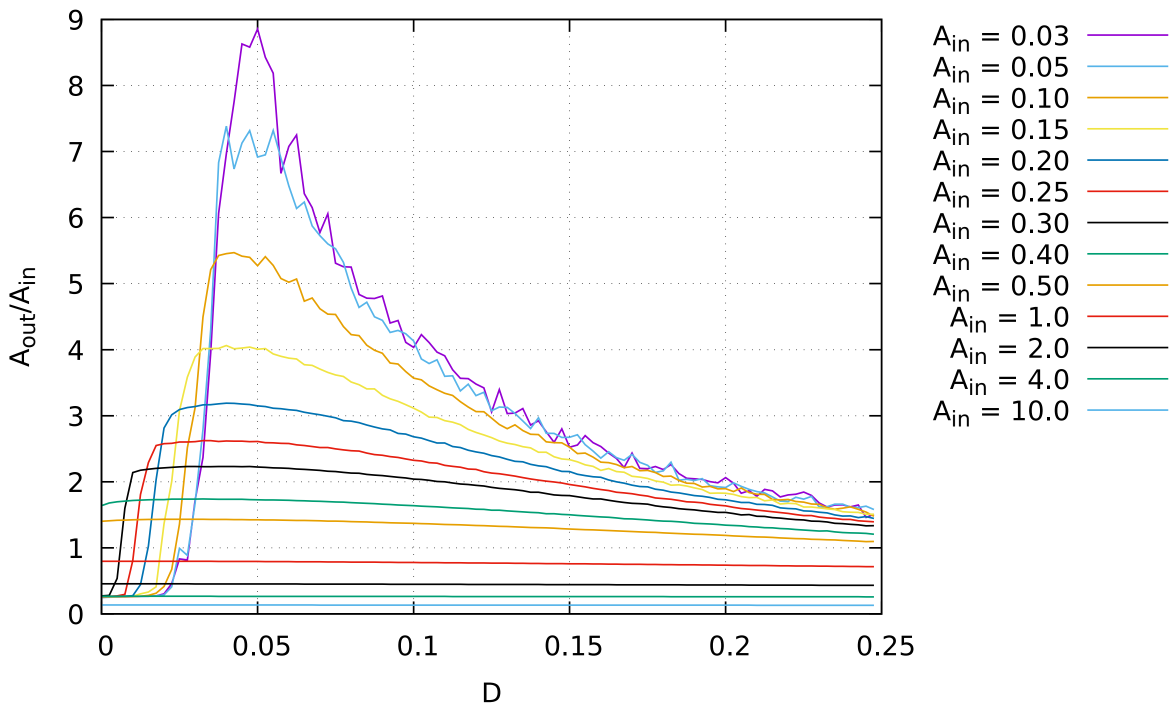


Figure 8: The response amplitude of all last plots merged in one plot. The maximum decreases and shifts to the left with the increasing input amplitude

At next we analyze the relation between the maximum of the output amplitude and the input amplitude see figure 9. We fitting the values to a inverse function and get

$$A_{\max}(A_{\text{in}}) \approx \frac{0.864292}{(A_{\text{in}} + 0.066830)} \tag{14}$$

As we saw before, it decreases with increasing of the input amplitude. Hence we get a smaller response by a higher input amplitude and the other way around.

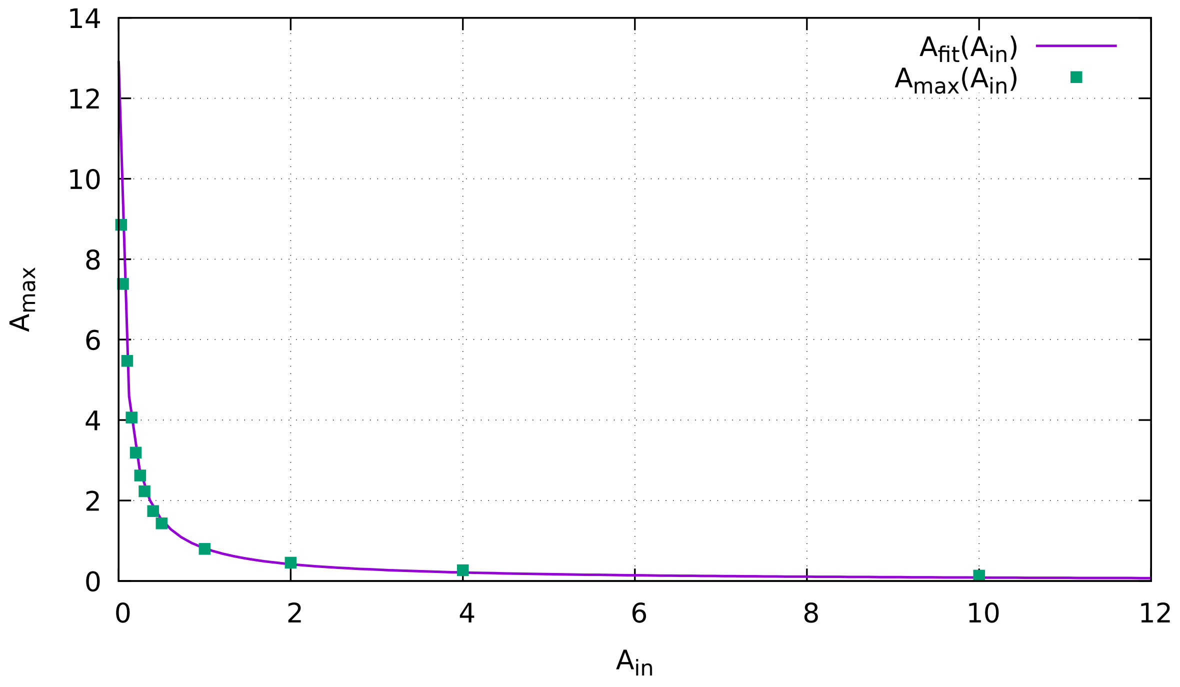


Figure 9: Green squares: maximum values of the output amplitudes. purple line: fit function of the maximum values. The inverse function suits to our maximum values.

3.5 Dependency of the time step Δt

In this subsection we want to analyze the simulations behavior on different time steps. Recap chapter 2.1 and equation 7:

$$x_{i+1} = x_i + (x_i - x_i^3 + A\cos(\Omega t_i))\Delta t_i + z_i\sqrt{2D\Delta t_i}$$

As one can see, the statistical part (i.e. noise strength) is dependent on the time step Δt, whereas the deterministic part is not. That will result in different behavior regarding different time steps. We observed divergent behavior if the time steps are of the size of 0,1s to 1s. This can be explained as follows: If the deflection (caused by noise) gets very high, the potentials backward driving force is also very high. If now the time step is also relatively big, the next iteration would yield a position result on the other side of the potential, ignoring the two stable stats. Now the deflection is even higher then before, just on the other side. The change in potential to lower values (i.e. the stable states) is ignored, because of the linear approximation through numeric iteration. The next iteration is driven backward again but with an even greater elongated position. After several steps, the position reaches infinity(in terms of data storage) and the simulation produces errors.

4 Summary

In our report we investigated the stochastic DGL of a random walking particle in a potential. We computed the solution with the Euler-Maruyama method. In that constellation, we validate the Kramer formula for a specific range of parameters. Than we switched on a oscillating force and we checked the phenomenon of stochastic resonance. In that case we found a relation of the output amplitude and input noise strength. We saw, that the output amplitude is here a nonlinear variable, because the maxima of amplitude function are varying with the input amplitude. All that results depend on the preset time step, because it regulates the accuracy of the computed values and it influences the strength of noise by a factor of $\sqrt{\Delta t}$.

Appendices

A Code

Listing 1: Quellcode `Main.cpp`

```
/*
Computational Physics Project
stochastical resonance

Alexander Putz 763265
Christian Gößl 762627

*/

#include <iostream>
#include <math.h>
#include <algorithm>
#include <string>
#include <sstream>
#include <fstream>
#include <random>
#include <ctime>
#include <cstdlib>
#include <vector>

using namespace std;

//declaration of used variables

default_random_engine gen;                                    //random
    number generators (gauss distribution)
normal_distribution<double> gauss(0, 1.0);       // gauss(mean, standard
    deviation)

ofstream stream; //file outputstream
ofstream stream_T_K;  // T_K file outputstream
ofstream stream_a_k; //for writing a_k
ofstream stream_b_k; //for writing b_k
ofstream stream_ampl_D;
ofstream stream_A_phi;

double tstep;   //steps for simulation
double M_PII = 3.14159265358979323846;
double A;
double omega;
double D;
double T;
double L;
double N;

int averages = 10; // std=10            number of averagings for Ampl-D
    relation
int tempdown = 0;
int tempup = 0;
double ampl_init;
double ampl_end;
double D_init;
double D_end;
double delta_a = 0;
double T_K0; // original Kramers time scale
double T_K; // Kramers time scale calculated with the frequency of the
    induced oscillation
bool mess = false; // indicator of states up and down
bool switchi = true; // switch for output yes or no
vector<long double> x;  //dynamic x Array
vector<long double> T_Kdown; // counts the down states
vector<long double> T_Kup; // count sthe up states
long double number;
// control of output of datapoints
double n_of_datapoints_to_file = 200;   //number of datapoints that shall
    be written to file
double GaussNoise(double mu, double q)
{
   static const double epsilon = std::numeric_limits<double>::min();
   static const double two_pi = 2.0*3.14159265358979323846;

   thread_local double z1;
   thread_local bool generate;
   generate = !generate;

   if (!generate)
      return z1 * q + mu;

   double u1, u2;
   do
   {
      u1 = rand() * (1.0 / RAND_MAX);
      u2 = rand() * (1.0 / RAND_MAX);
   } while (u1 <= epsilon);

   double z0;
   z0 = sqrt(-2.0 * log(u1)) * cos(two_pi * u2);
   z1 = sqrt(-2.0 * log(u1)) * sin(two_pi * u2);
   return z0 * q + mu;
}
void close_all_streams()
{
   stream.close();
   stream_T_K.close();
   stream_a_k.close();
   stream_b_k.close();
   stream_ampl_D.close();
   stream_A_phi.close();
}

void init(int tempL, double tempA, double tempomega, double tempD, int
    tempN, bool tempswitchi) {
   close_all_streams();
   srand(time(NULL));  // seed the random number generator with time
   A = tempA; //
   omega = tempomega; //
   D = tempD; //
   N = tempN; //
   L = tempL;
   tstep = double(L * 2 * M_PIl / (N * omega));
   switchi = tempswitchi;
   //stream.open("data_D" + to_string(D) + ".txt");
   stream.open("data.txt");
   stream_T_K.open("T_K.txt", ios::app);
   stream_a_k.open("a_k_omega" + to_string(omega) + ".txt", ios::app);
```

```cpp
    stream_b_k.open("b_k_omega" + to_string(omega) + ".txt", ios::app);
    stream_A_phi.open("A_k-phi_k_omega" + to_string(omega) + ".txt", ios::app);
    stream_ampl_D.open("ampl_D.txt", ios::app);
    if (stream.is_open()) {
        stream << "# t    x        gauss" << endl; //setting header lines
    }
    else cout << "ERROR opening file!" << endl; //setting error
    number = GaussNoise(0, 1);//
}

struct FFT_a_b {
    double a_k;            //cos Fourier components
    double b_k;            //sin Fourier components
    double A_k;      // Amplitude of a_k and b_k
    double phi_k;    // Phase of a_k and b_k
};
FFT_a_b FFT(vector<long double> x)
{
    FFT_a_b coefficients;
    long double help = 0.;

    help = 0.;
    for (int i = 0; i < N; i++) //integrate Fourier integral
    {
        help += x[i] * cos(omega*i*tstep);
    }
    help *= 1.0 / double(N);

    coefficients.a_k = help; //add to the list

    help = 0.;    //reset for b_k integration
    for (int i = 0; i < N; i++)
    {
        help += x[i] * sin(omega*i*tstep);
    }
    help *= 1.0 / double(N);

    coefficients.b_k = help; //add to list
    coefficients.A_k = sqrt(pow(coefficients.a_k, 2) + pow(coefficients.b_k, 2));
    coefficients.phi_k = atan(coefficients.a_k / coefficients.b_k);
    return coefficients;
}
void Output(int file, long double data1, long double data2, long double data3 = 0, long double data4 = 0, long double data5 = 0)         //data 3, 4 and 5 are optional
{
    //  t    x gauss
    if (file == 0) stream << data1 << "  " << data2 << " " << data3 << endl;
    //  k    a_k
    if (file == 1) stream_a_k << data1 << "      " << data2 << endl;
    //  k    b_k
    if (file == 2) stream_b_k << data1 << "      " << data2 << endl;
    //  D    A_k    phi_k
    if (file == 3) stream_A_phi << data1 << "     " << data2 << "  " << data3 << endl;
    //  ampl
    if (file == 4) stream_ampl_D << data1 << "   " << data2 << " " << data3 << endl;

    if (file == 5) stream_A_phi << data1 << "     " << data2 << " " << data3
        << " " << data4 << " " << data5 << endl;
}
long double NumCalc(long double x, long double number, int i) //
    stochastical differential equation
{
    return x + (x - pow(x, 3) + A * cos(omega*double(i)*tstep))*tstep +
        number*sqrt(2 * D * tstep);
}
void kramer(int k, double mean)
{
    int j = k;
    if ((x.at(j) < -0.99) && (mess == false))
    {
        tempdown = j;
        mess = true;
        if ((tempup > 1) || (mean > 0))
        {
            T_Kup.push_back(double(j - tempup)*tstep);
        }
    }
    if ( (x.at(j) > 0.99) && (mess == true) )
    {
        T_Kdown.push_back(double(j - tempdown)*tstep);
        tempup = j;
        mess = false;
    }
    if (j + 1 == N)
    {
        if (mess == true)
            T_Kdown.push_back(double(j - tempdown)*tstep);
        else
            T_Kup.push_back(double(j - tempup)*tstep);
    }
}
void set_Data(double L, double A, double omega, double D, double N)
{
    //L, A, omega, D, N
    init(L, A, omega, D, N, switchi);
    stream << "# parameter(L, A, omega, D, N, tstep): " << L << ", " << A <<
         ", " << omega << ", " << D_init << ", " << D_end << ", " << N <<  ",
         " << tstep <<endl;
    Output(0, 0., x.at(0), number);

    for (int i = 0; i < N; i++) {
        x.push_back(NumCalc(x.at(i), number, i)); //
        if ((i%int(n_of_datapoints_to_file) == 0) and (switchi==true))//write
             every "reduced" step to data, write only 100 steps to file
          Output(0, double(i + 1)*tstep, x[i + 1], number);        // write
             calculated data to file
        number = GaussNoise(0,1);//
    }
}
void docu()
{
    set_Data(30, 0, 0.5, 0.4, 1E8);
    switchi=true;
    ampl_init = A;
    ampl_end = 0.5;
    D_init = D;
    D_end = 0.5;
```

```
}
void Kramer_analysis()
{
    remove("T_K.txt");
    docu();
    stream_T_K << "# parameter(L, A, omega, D_init, D_end, N, tstep): " << L
         << ", " << A << ", " << omega << ", " << D_init << ", " << D_end <<
        ", " << N << ", " << tstep <<endl;
    for (double D = D_init; D <= D_end; D = D + 0.02)
    {
        double mean_T_K_size = 0;
        int M = 1;
        for (int m = 1; m <= M; m++) // how many times of repititions
        {
            double mean = 0;
            for (int i = 0; i < N; i++)
            {
                mean = x[i] + mean;
                kramer(i, mean);
            }
            mean_T_K_size = mean_T_K_size + (T_Kup.size() + T_Kdown.size());
            x.clear();
            T_Kup.clear();
            T_Kdown.clear();
            x.push_back(0);
            set_Data(L, A, omega, D, N);
        }
        stream_T_K << (1/D) << " " << log(N*tstep / mean_T_K_size*M) << endl;
    }
}
double return_variance(vector <double> data, double avg) {
    double deviation = 0;

    for (int i = 0; i < data.size(); i++) {        //calculate stochastic
        variance
        deviation += pow((data[i] - avg), 2);
    }

    return deviation / data.size();

}
void Fourier_task()
{
    double amp_avg = 0;
    double phi_avg = 0;
    vector <double> amp_vec;
    vector <double> phi_vec;


    double amp_dev = 0;
    double phi_dev = 0;

    for (double D = 0; D <= 0.25; D += 0.0025)
    {
        cout << " at D=" << D;
        for (int i = 0; i < averages; i++) {        // averages- Iterations to
            get statistical data

                                        //L, A, omega, D, N
            set_Data(45, 0.1, 0.001, D, 2E6);

```

```cpp
            FFT_a_b coefficients = FFT(x);

            amp_avg += coefficients.A_k;            // sum up all amplitudes and phi to get their average value
            phi_avg += coefficients.phi_k;

            amp_vec.push_back(coefficients.A_k);    // store values in a vector/array to be able to calculate variance
            phi_vec.push_back(coefficients.phi_k);

            //for troubleshooting
            if (i == 1) {
                cout << "D= " << D << endl;
                cout << "a_k= " << coefficients.a_k << endl;
                cout << "b_k= " << coefficients.b_k << endl;

            }
            x.clear();
            x.push_back(0);
        }
        amp_avg /= averages;
        cout << "  amp_avg = " << amp_avg<<endl;
        phi_avg /= averages;

        //get deviation out of amp_vec
        amp_dev = return_variance(amp_vec, amp_avg);
        phi_dev = return_variance(phi_vec, phi_avg);

        Output(5, D, amp_avg, phi_avg, amp_dev, phi_dev);
        amp_avg = 0;
        phi_avg = 0;
        amp_dev = 0;
        phi_dev = 0;

    }
}
int main(int argc, char* argv[]) {

    x.push_back(0);

    bool only_FF = false;
    if (only_FF == true)

        Fourier_task();

    else Kramer_analysis();

    close_all_streams();

    return 0;
}
```

References

1. Luca Gammaitoni, Peter Hänggi, Peter Jung, and Fabio Marchesoni. Stochastic resonance. 70(1):223–287. doi: 10.1103/RevModPhys.70.223. URL https://link.aps.org/doi/10.1103/RevModPhys.70.223.

2. Catherine Rouvas-Nicolis and Gregoire Nicolis. Stochastic resonance - scholarpedia. 2(11):1474. ISSN 1941-6016. doi: 10.4249/scholarpedia.1474. URL http://www.scholarpedia.org/article/Stochastic_resonance.

3. Timothy Sauer. Numerical solution of stochastic differential equations in finance. In Jin-Chuan Duan, Wolfgang Karl Härdle, and James E. Gentle, editors, *Handbook of Computational Finance*, pages 529–550. Springer Berlin Heidelberg. ISBN 978-3-642-17253-3 978-3-642-17254-0. doi: 10.1007/978-3-642-17254-0_19. URL http://link.springer.com/10.1007/978-3-642-17254-0_19.

4. B. K. Øksendal. *Stochastic differential equations: an introduction with applications*. Universitext. Springer, 6th ed., corr. 4th print edition. ISBN 978-3-540-04758-2. OCLC: ocn166267310.