/Free-Diffusion/main.py (used for 7.1)

```python
#!/usr/bin/python

import numpy as np

import matplotlib
matplotlib.use("tkagg")

from matplotlib import pyplot as plt

if __name__ == "__main__":
    x0 = 0
    sigma = 1.0
    dt = 1.0
    dt_sqrt = np.sqrt(dt)

    N = 10000
    T = 1000

    x = np.zeros((N,int(T/dt)))
    x[:,0] = x0

    t = 0.0
    j = 0

    while t < T-1:
        diff = np.round(np.random.random_sample((1,N))) * 2 - 1
        x[:,j+1] = x[:,j] + diff * sigma * dt_sqrt

        t += dt
        j += 1

    BINS = 31
    LWR_BND = -j*sigma*dt_sqrt / 10
    UPR_BND = j*sigma*dt_sqrt / 10
    counts0, bins0 = np.histogram(x[:,j//10], bins=BINS, range=(LWR_BND,UPR_BND))
    counts1, bins1 = np.histogram(x[:,j//2], bins=BINS, range=(LWR_BND,UPR_BND))
    counts2, bins2 = np.histogram(x[:,j], bins=BINS, range=(LWR_BND,UPR_BND))

    print(f"j={j//10}")
    print(f"  Avg:{np.mean(x[:,j//10])} Std:{np.std(x[:,j//10])} Expected std:{sigma*np.sqrt(j//10*dt)}")
    print(f"j={j//2}")
    print(f"  Avg:{np.mean(x[:,j//2])} Std:{np.std(x[:,j//2])} Expected std:{sigma*np.sqrt(j//2*dt)}")
    print(f"j={j}")
    print(f"  Avg:{np.mean(x[:,j])} Std:{np.std(x[:,j])} Expected std:{sigma*np.sqrt(j*dt)}")

    plt.stairs(counts0,bins0, fill=True, alpha=0.2, color="#590995", label=f"$j={j//10}$")
    plt.stairs(counts1,bins1, fill=True, alpha=0.2, color="#03C4A1", label=f"$j={j//2}$")
    plt.stairs(counts2,bins2, fill=True, alpha=0.2, color="#C62A88", label=f"$j={j}$")
    plt.stairs(counts0,bins0, color="#590995")
    plt.stairs(counts1,bins1, color="#03C4A1")
    plt.stairs(counts2,bins2, color="#C62A88")

    plt.xlabel("$x_j$")
    plt.ylabel("Count")
    plt.legend()
    plt.grid()

    plt.show()
```

/Diffusion-In-Box/src/main.c (used for 7.2)

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

const double x0 = 0;
const double sigma = 1;
const double dt = 0.01;

const int N = 10000;
const double T = 10;
const double L = 100;

void dumpCsv(FILE* fptr, double* x) {
    for ( int i = 0; i < N; i++ ) {
        fprintf(fptr, ”%f,”, x[i]);
    }
    fprintf(fptr, ”\n”);
}

void setDirs(float* dirs) {
    for ( int i = 0; i < N; i++ ) {
        dirs[i] = rand() % 2 * 2 - 1;
    }
}

void getNowAsDateStr(char* text, int textLen) {
    time_t now = time(NULL);
    struct tm *t = localtime(&now);

    /* Leave one byte for null-termination */
    strftime(text, textLen-1, ”./output/%d_%m_%Y_%H-%M-%S.csv”, t);
}

void simulateTrajectories(double* x, int n) {
    const double dt_sqrt = sqrt(dt);

    for ( int i = 0; i < n; i++ ) {
        x[i] = x0;
    }

    double t = 0;
    int j = 0;

    srand(time(NULL));
    float diff[n];

    while ( t < T ) {
        setDirs(diff);
        for ( int i = 0; i < n; i++ ) {
            diff[i] *= sigma * dt_sqrt;
            x[i] += diff[i];

            if ( x[i] < -L/2 ) {
                x[i] = -L - x[i];
            } else if ( x[i] > L/2 ) {
                x[i] = L - x[i];
            }
        }

        if ( j % 10000 == 0 ) {
            printf(”j: %d, t: %f/%f\n”, j, t, T);
        }

        t += dt;
        j++;
    }
}

int main() {
    double x[N];

    simulateTrajectories(x, N);

    char dateString[100];
    getNowAsDateStr(dateString, sizeof(dateString));

    FILE* fptr;
    fptr = fopen(dateString, ”w”);

    dumpCsv(fptr, x);

    return 0;
}
```

/Multiplicative-Noise/src/main.c (used for 7.3)

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

const double x0 = 0;
const double dt = 0.01;
const double sigma0 = 1;
const double delta_sigma = 1.8;

const int N = 10000;
const double T = 10;
const double L = 100;

void dumpCsv(FILE* fptr, double* x) {
    for ( int i = 0; i < N; i++ ) {
        fprintf(fptr, ”%f,”, x[i]);
    }
    fprintf(fptr, ”\n”);
}

void setDirs(float* dirs) {
    for ( int i = 0; i < N; i++ ) {
        dirs[i] = rand() % 2 * 2 - 1;
    }
}

void getNowAsDateStr(char* text, int textLen) {
    time_t now = time(NULL);
    struct tm *t = localtime(&now);

    /* Leave one byte for null-termination */
    strftime(text, textLen-1, ”./output/%d_%m_%Y_%H-%M-%S.csv”, t);
}

double sigma(double x) {
    return sigma0 + delta_sigma/L * x;
}

void simulateTrajectories(double* x, int n) {
    const double dt_sqrt = sqrt(dt);

    for ( int i = 0; i < n; i++ ) {
        x[i] = x0;
    }

    double t = 0;
    int j = 0;

    srand(time(NULL));
    float diff[n];

    while ( t < T ) {
        setDirs(diff);
        for ( int i = 0; i < n; i++ ) {
            diff[i] *= sigma(x[i]) * dt_sqrt;
            x[i] += diff[i];

            if ( x[i] < -L/2 ) {
                x[i] = -L - x[i];
            } else if ( x[i] > L/2 ) {
                x[i] = L - x[i];
            }
        }

        if ( j % 10000 == 0 ) {
            printf(”j: %d, t: %f/%f\n”, j, t, T);
        }

        t += dt;
        j++;
    }
}

int main() {
    double x[N];

    simulateTrajectories(x, N);

    char dateString[100];
    getNowAsDateStr(dateString, sizeof(dateString));

    FILE* fptr;
    fptr = fopen(dateString, ”w”);

    dumpCsv(fptr, x);

    return 0;
}
```

/Spurious-Drift/src/main.c (used for 7.4)

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

const double x0 = 0;
const double dt = 0.01;
const double sigma0 = 1;
const double delta_sigma = 1.8;

const int N = 10000;
const double T = 100;
const double L = 100;
const double ALPHA = 0.5;

void dumpCsv(FILE* fptr, double* x) {
    for ( int i = 0; i < N; i++ ) {
        fprintf(fptr, "%f,", x[i]);
    }
    fprintf(fptr, "\n");
}

void setDirs(float* dirs) {
    for ( int i = 0; i < N; i++ ) {
        dirs[i] = rand() % 2 * 2 - 1;
    }
}

void getNowAsDateStr(char* text, int textLen) {
    time_t now = time(NULL);
    struct tm *t = localtime(&now);

    /* Leave one byte for null-termination */
    strftime(text, textLen-1, "./output/%d_%m_%Y_%H-%M-%S.csv", t);
}

double sigma(double x) {
    return sigma0 + delta_sigma/L * x;
}

double dsigma(double x) {
    return delta_sigma/L;
}

void simulateTrajectories(double* x, int n) {
    const double dt_sqrt = sqrt(dt);

    for ( int i = 0; i < n; i++ ) {
        x[i] = x0;
    }

    double t = 0;
    int j = 0;

    srand(time(NULL));
    float diff[n];

    while ( t < T ) {
        setDirs(diff);
        for ( int i = 0; i < n; i++ ) {
            diff[i] *= sigma(x[i]) * dt_sqrt;

            /* Noise-induced drift */
            x[i] += ALPHA * sigma(x[i]) * dsigma(x[i]) * dt;

            x[i] += diff[i];
            if ( x[i] < -L/2 ) {
                x[i] = -L - x[i];
            } else if ( x[i] > L/2 ) {
                x[i] = L - x[i];
            }
        }

        if ( j % 10000 == 0 ) {
            printf(”j: %d, t: %f/%f\n”, j, t, T);
        }

        t += dt;
        j++;
    }
}

int main() {
    double x[N];

    simulateTrajectories(x, N);

    char dateString[100];
    getNowAsDateStr(dateString, sizeof(dateString));

    FILE* fptr;
    fptr = fopen(dateString, ”w”);

    dumpCsv(fptr, x);

    return 0;
}
```

/[Diffusion-In-Box|Multiplicative-Noise|Spurious-Drift]/main.py (used for 7.2, 7.3, 7.4)

```
#!/usr/bin/python

import numpy as np
import matplotlib
matplotlib.use("tkagg")

from matplotlib import pyplot as plt

BINS = 100
L = 100
LWR_BND = -L/2
UPR_BND = L/2

if __name__ == "__main__":

    for color, t in zip(["#537c78", "#cc222b", "#f15b4c", "#faa41b", "#ffd45b"],[10, 100,
        1000, 10000, 100000]):
        with open(f"./output/{t}.csv") as f:
            contents = f.read()
            x = np.array(list(map(float, contents.split(",")[:-1])))

        counts, bins = np.histogram(x, bins=BINS, range=(LWR_BND,UPR_BND))
        print(f"  Avg:{np.mean(x[:])} Std:{np.std(x[:]) / L}")
        plt.stairs(counts,bins, fill=True, alpha=0.2, color=color, label=f"$t={t} s$")
        plt.stairs(counts,bins, color=color)


    plt.xlabel("$x_j$")
    plt.ylabel("Count")
    plt.legend()
    plt.grid()

    plt.show()
```