Міністерство освіти і науки України
Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЗВІТ

з лабораторної роботи №2.2 з дисципліни
«Інтелектуальні вбудовані системи»

Виконав:
ІП-83
Сергійчук Н. С.

Київ 2020

1. Завдання

Для згенерованого випадкового сигналу з Лабораторної роботи N 1 відповідно до заданого варіантом (Додаток 1) побудувати його спектр, використовуючи процедуру швидкого перетворення Фур'є з проріджуванням відліків сигналу за часом. Розробити відповідну програму і вивести отримані значення і графіки відповідних параметрів

2. Лістинг програми

```
fn is_power_of_two(x: usize) -> bool {
(x & (x - 1)) == 0
}

fn fft_inner(buf_a: &mut [Complex<f64>], buf_b: &mut [Complex<f64>], n:
usize, step: usize) {
const I: Complex<f64> = Complex { re: 0.0, im: 1.0 };

if step >= n {
return;
}

fft_inner(buf_b, buf_a, n, step * 2);
fft_inner(&mut buf_b[step..], &mut buf_a[step..], n, step * 2);
// create a slice for each half of buf_a:
let (left, right) = buf_a.split_at_mut(n / 2);

for i in (0..n).step_by(step * 2) {
let t = (-I * PI * (i as f64) / (n as f64)).exp() * buf_b[i + step];
left[i / 2] = buf_b[i] + t;
right[i / 2] = buf_b[i] - t;
```

```
}

}
```

```
pub fn fft(signal: &[f64]) -> Vec<f64> {

let n = signal.len();

assert!(is_power_of_two(n));
```

```
let mut buf_a: Vec<_> = signal.iter().map(Complex::from).collect();

// alternate between buf_a and buf_b to avoid allocating a new vector
each time:

let mut buf_b = buf_a.clone();

fft_inner(&mut buf_a, &mut buf_b, n, 1);

buf_a.into_iter().map(Complex::norm).collect()

}
```

```
use core::f64;

use fourier::fft;

use num::Complex;

use rustfft::FftPlanner;

use signal::gen_signal;

use std::io::Write;

use std::{fmt::Display, fs::File};


const HARMONICS: usize = 12;

const FREQUENCY: usize = 900;

const INTERVALS: usize = 256;

const DT: f64 = 1.0;
```

```
const USAGE: &str = "Usage: <signal.dat> <fourier.dat>";
```

```
fn save<T: Write, D: Display>(mut dest: T, slice: &[D], header: &str,
mult: f64) {
writeln!(dest, "{}", header).unwrap();
for (i, s) in slice.iter().enumerate() {
writeln!(dest, "{}\t{}", i as f64 * mult, s).unwrap();
}
}
```

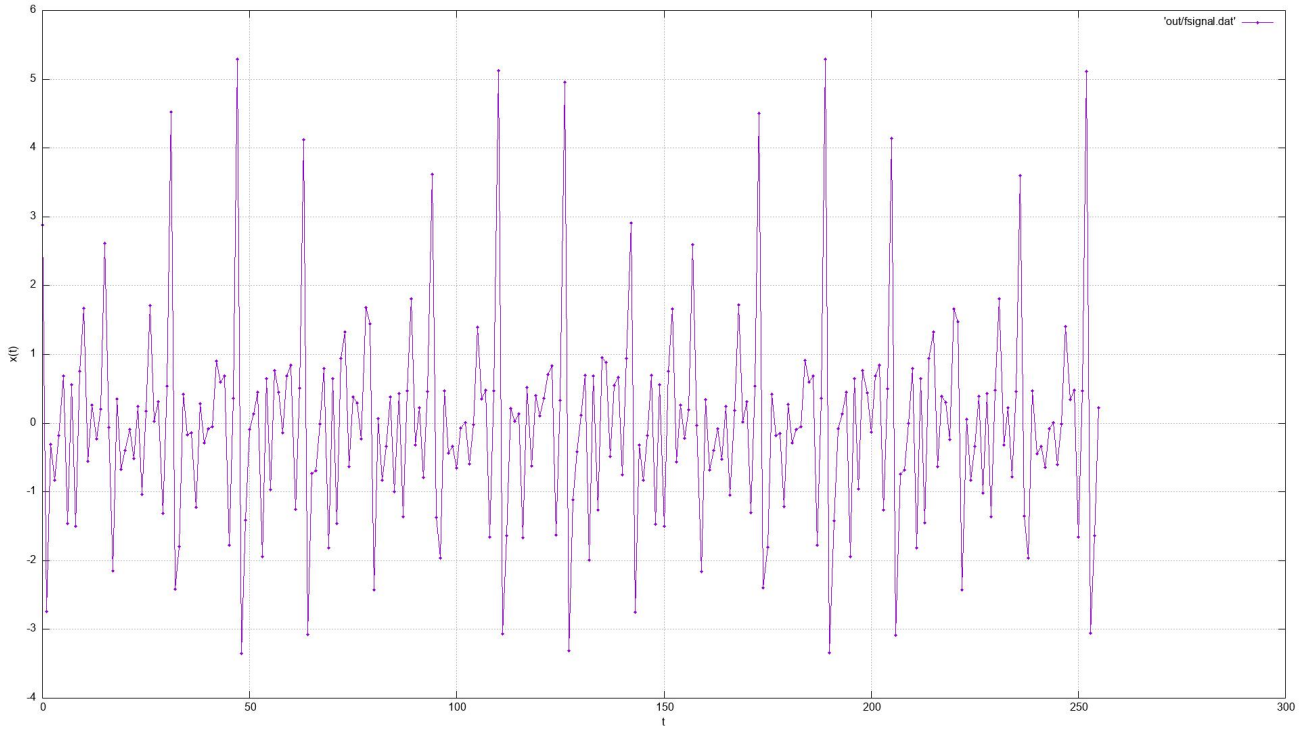
```
fn lib_fft(signal: &[f64]) -> Vec<f64> {
let mut planner = FftPlanner::new();
let fft = planner.plan_fft_forward(signal.len());
```

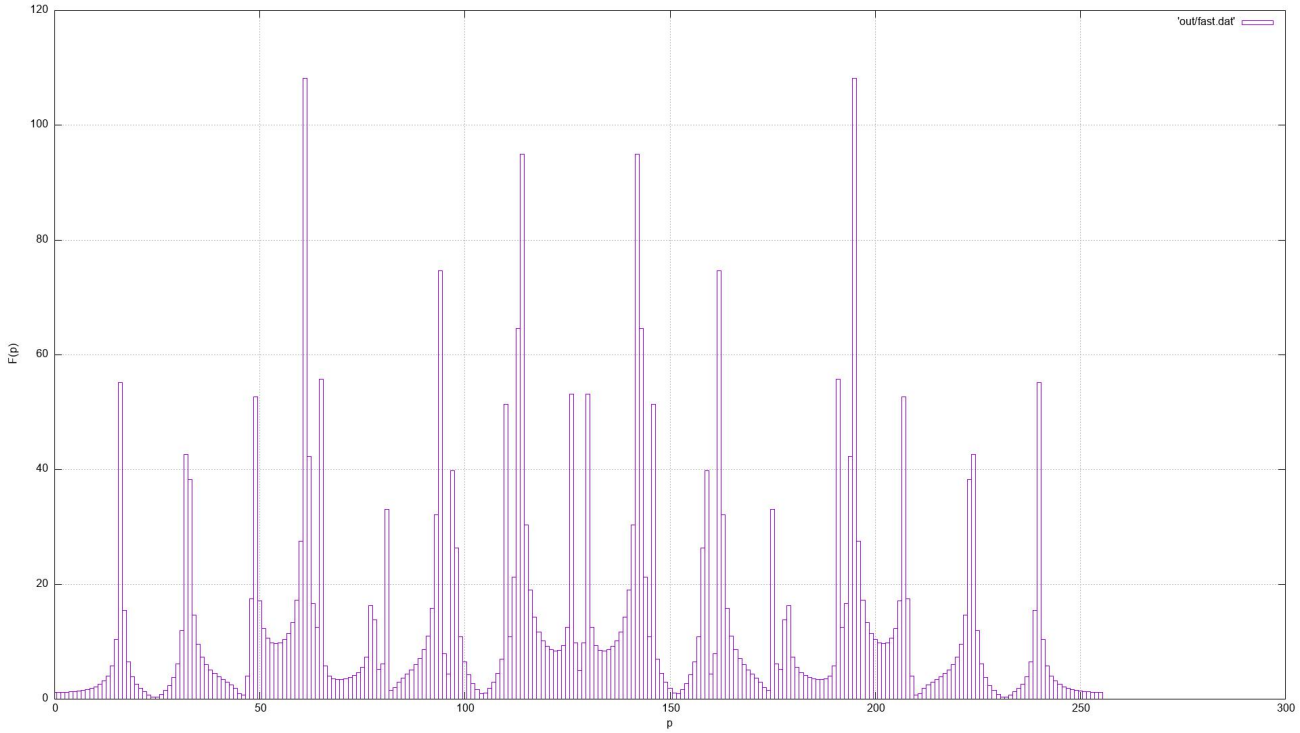
```
let mut buffer: Vec<_> = signal.iter().map(Complex::from).collect();
fft.process(&mut buffer);
buffer.into_iter().map(Complex::norm).collect()
}
```

```
fn almost_eq(a: &[f64], b: &[f64]) -> bool {
const EPS: f64 = 0.0000000001;
if a.len() != b.len() {
return false;
}
!a.iter().zip(b.iter()).any(|(a, b)| (a - b).abs() > EPS)
}
```

```rust
fn main() {
let signal_dest = std::env::args().nth(1).expect(USAGE);
let fourier_dest = std::env::args().nth(2).expect(USAGE);

let signal_dest = File::create(signal_dest).unwrap();
let fourier_dest = File::create(fourier_dest).unwrap();

let signal = gen_signal(HARMONICS, FREQUENCY, INTERVALS, DT);
save(signal_dest, &signal, "# t\tx(t)", DT);

let fourier = fft(&signal);
save(fourier_dest, &fourier, "# p\tX(p)", DT);

// Compare with library
let lib = lib_fft(&signal);
assert!(almost_eq(&lib, &fourier));
}
```

3. Результати виконання кожної програми.


'out/fsignal.dat'
x(t)
t
6
5
4
3
2
1
0
-1
-2
-3
-4
0
50
100
150
200
250
300


'out/fast.dat'
F(p)
p
120
100
80
60
40
20
0
0
50
100
150
200
250
300