Нахождение неизвестной синусоиды

Добрый вечер, сегодня мы займемся нахождением параметров входного *синусоидального сигнала* большой частоты, когда обычный метод построения по точкам нам не подходит, хотя получилось, что всё равно можно. Было решено не использовать интерполяцию, разложение в ряд Тейлора, Маклорена, преобразования Фурье для того чтобы доказать, что простыми методами "в лоб" можно тоже найти все параметры. Сначала мы разработаем код для иллюстрации метода на языке **Python**, а потом реализуем метод на языке C++ для МК Arduino Uno.
К ардуино подключается синусоидальный сигнал ко входу A0.

Условия задачи

- На вход АЦП подается синусоидальный сигнал
- Нужно вывести на экран компьютера значение A_0, ω_0, φ_0
- Считывать значения сигнала можно с частотой не ниже частоты самого АЦП

Частоты АЦП

```
Микроконтроллер Atmega 328 с тактовой частотой 16 МГц, используемые в Arduino Uno, Nano, Pro Mini, и  микроконтроллер ATmega2560 используемый в Arduino MEGA 2560, содержат шестиканальный АЦП, разрядность которого составляет 10 бит. Считывание значение с аналогового порта занимает 100 микросекунд (0.0001 сек), получается, что максимальная частота считывания приблизительно 10,000 Гц.
```

Согласно сайту Codius:

```
Здесь стоит подробнее рассказать о том, как значения битов ADPS[2:0] влияют на скорость и точность АЦП. Частота работы МК Atmega 328P 16МГц. При настройках по умолчанию используется предделитель 128 (ADPS[2:0]=[111]), а это значит, что АЦП работает на частоте 16МГц/128=125КГц, что укладывается в данные даташита – 50-200КГц. Отсюда очень низкая скорость выполнения преобразования, но и самая высокая точность. Для того, чтобы ускорить работу АЦП необходимо уменьшить предделитель, но необходимо помнить, что чем выше частота, тем ниже точность преобразования. Экспериментально можно получить значение предделителя – 16 (ADPS[2:0]=[100]), при котором возможен компромисс 10-кратного прироста скорости, при сохранении точности.
```

ADPS[2:0]	Коэф. деления
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Согласно сайту ACDC.foxylab:

```
Установка множителя, равного 16 (частота дискретизации около 76 кГц) -
sbi(ADCSRA,ADPS2) ;
cbi(ADCSRA,ADPS1) ;
cbi(ADCSRA,ADPS0) ;
В этом случае тактовая частота АЦП равна 1 МГц. С учетом того, что
преобразование занимает 13 тактов, оно будет длиться 13 мкс (частота 1000/13 =
76,92 мкс). Но это в идеальном случае! Тесты показали, что без изменения
множителя (128 по умолчанию) одно аналого-цифровое преобразование (analogRead) в
цикле занимает ~ 112 мкс, а с измененным множителем (16) - ~ 17 мкс.
```

Решение задачи

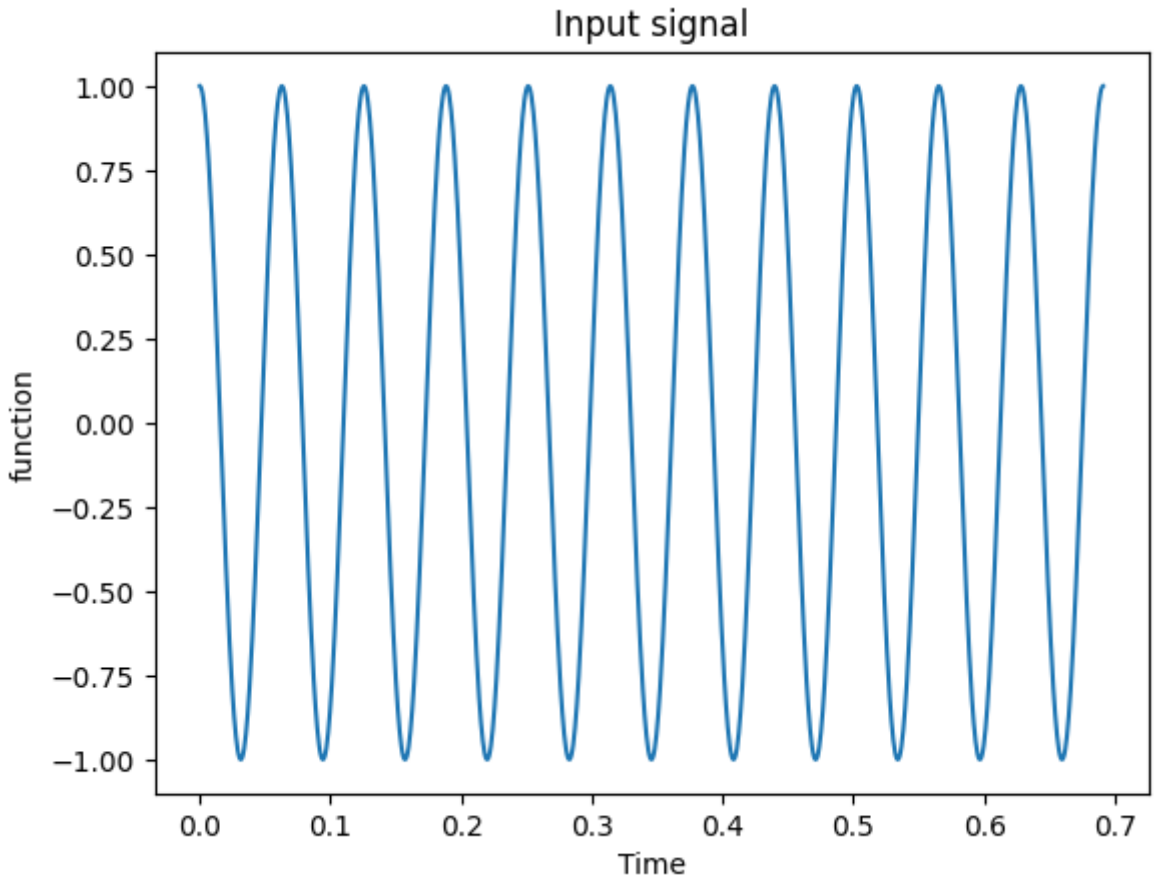
Для начала смоделируем реальные условия, для этого нам нужно построить массив данных о каком-то синусе, чтобы построить его для наглядности

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math
import random
a0=1
w0=100
phi0=math.pi/2
print("Программа для нахождения неизвестной синусоиды\n-----------------------")
class Function:
    a0=1
    w0=100
```

```
    phi0=math.pi/2

    def sin_t(self, q):
        return self.a0*math.sin(self.w0*q+phi0)

print(f"Параметры теор синуса:A={a0}, w={w0},phi={phi0}")

num=1000
end=22*math.pi*w0**(-1)
step=end/num

print(f"Шаг точек синуса:{step}")

t = np.linspace(0, end, num=num)
def sin_0(a0:float,
          w0:float,
          phi0:float,
          t:int):
    f0=[]

    for i in range(num):
        f0.append(a0*math.sin(w0*t[i]+phi0))

    return f0

#print(f0)
f0=sin_0(a0,w0,phi0,t)
plt.plot(t, f0)
plt.xlabel('Time')
plt.ylabel('function')
plt.title('Input signal')
'''
w=4
Fun=Function()
t2=[]
t1=[]
for r in range(0,30):
    t1.append(step*10)
    t2.append(Fun.sin_t(t1[r]))
print(t2)
plt.scatter(t1, t2, 40, 'g', 'o', alpha=0.8)
'''
```

```
Программа для нахождения неизвестной синусоиды
-----------------------
Параметры теор синуса:A=1, w=100,phi=1.5707963267948966
Шаг точек синуса:0.0006911503837897545
```

```
"\nw=4\nFun=Function()\nt2=[]\nt1=[]\nfor r in range(0,30):\n
t1.append(step*10)\n    t2.append(Fun.sin_t(t1[r]))\nprint(t2)\nplt.scatter(t1,
t2, 40, 'g', 'o', alpha=0.8)\n"
```



Теперь сигнал построен, и мы может начать его искать.

Идея метода

Решение строится на основе нахождения максимальных значений сигнала. Среди каждых N значений выбираются два максимальных значения, и записываются в небольшой массив вместе со своими значениями тактов. Тогда мы сможем перевести эти такты в пройденное время, тк используя команду millis() в МК сможет считать время на выполнение каждой команды. После этого, если мы знаем примерное значения частоты сигнала, то округляя следующую величину

$$period = round\left(\frac{(t_2 - t_1)\cdot\omega_0}{2\pi}\right)$$

Мы сможем найти количество пройденных периодов между этими значениями, а значит сможем вычислить точную частоту сигнала. об Амплитуде сигнала я вообще молчу, когда найдем максимальные точки сигнала, то вот она и амплитуда. С фазой уже проблемы есть, потому что нужно думать

Примечание

Для того, чтобы мы смогли таким методом найти параметры синусоиды, необходимо чтобы МК знал примерную частоту входного сигнала, чтобы потом мы сразу могли посчитать количество пройденных периодов между найденными максимумами

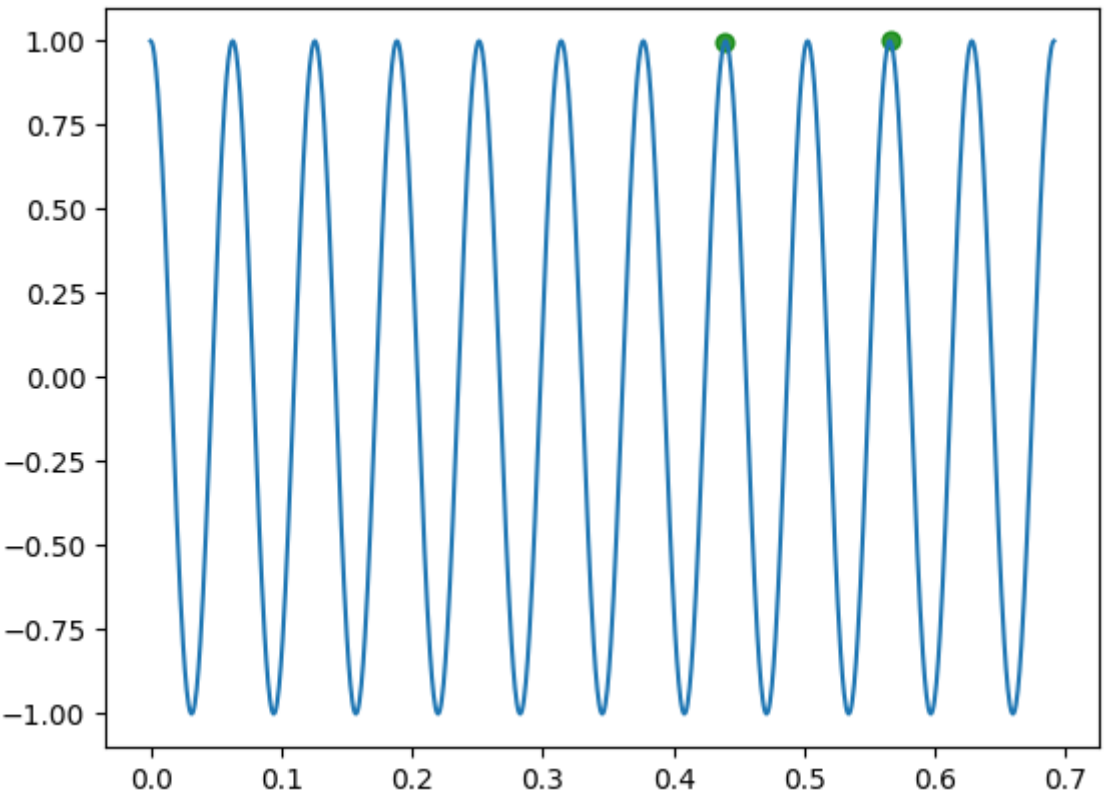
Реализация на python

Для упрощения я использовал нахождение num1=1000 случайных точек сигнала на промежутке уже готового графика, чтобы показать, что по нему можно искать параметры.
Каждый перезапуск блока дает нам новые точки, и, соответственно, новые результаты

```
t1=[]
t2=[]
Fun=Function()
plt.plot(t, f0)
num1=1000
end=end
w0=w0-1
w=w0/5
step1=end/num1
for r in range (num1):
    t1.append(step1*random.uniform(0, num1))
t1.sort()

for r in range (num1):
    t2.append(Fun.sin_t(t1[r]))
#print(len(t1),len(t2))
t2max=-100
t1m=[]
t2m=[]
t1m.append(1)
t1m.append(1)
t2m.append(-100)
t2m.append(-100)
for r in range (num1):
    if t2[r]>t2m[1]:
        t2m[0]=t2m[1]
        t1m[0]=t1m[1]
        t2m[1]=t2[r]
```

```
        t1m[1]=t1[r]
#print(t1m,t2m)
a1=max(t2m)
print(f"Нашли амплитуду, она примерно равна A={a1}\nПромежуток времени между
максимумами dt={abs(t1m[0]-t1m[1])}\nКоличество периодов между точками:
{(abs(t1m[0]-t1m[1]))*w0/(math.pi*2)}")
print(f"Целое число периодов тогда per={round((abs(t1m[0]-
t1m[1]))*w0/(math.pi*2))}")
per=round((abs(t1m[0]-t1m[1]))*w0/(math.pi*2))
w_check=per*math.pi*2/abs(t1m[0]-t1m[1])
print(f"Нашли частоту, она примерно равно w0={w_check}")
plt.scatter(t1m, t2m, 40, 'g', 'o', alpha=0.8)
while(t1m[0]>w0/(math.pi*2)):
    t1m[0]-=w0/(math.pi*2)
print(t1m[0])
```

```
Нашли амплитуду, она примерно равна A=0.9999990449504748
Промежуток времени между максимумами dt=0.12586516772814188
Количество периодов между точками: 1.68269334299004
Целое число периодов тогда per=2
Нашли частоту, она примерно равно w0=99.83993857221459
0.43963533055242937
```

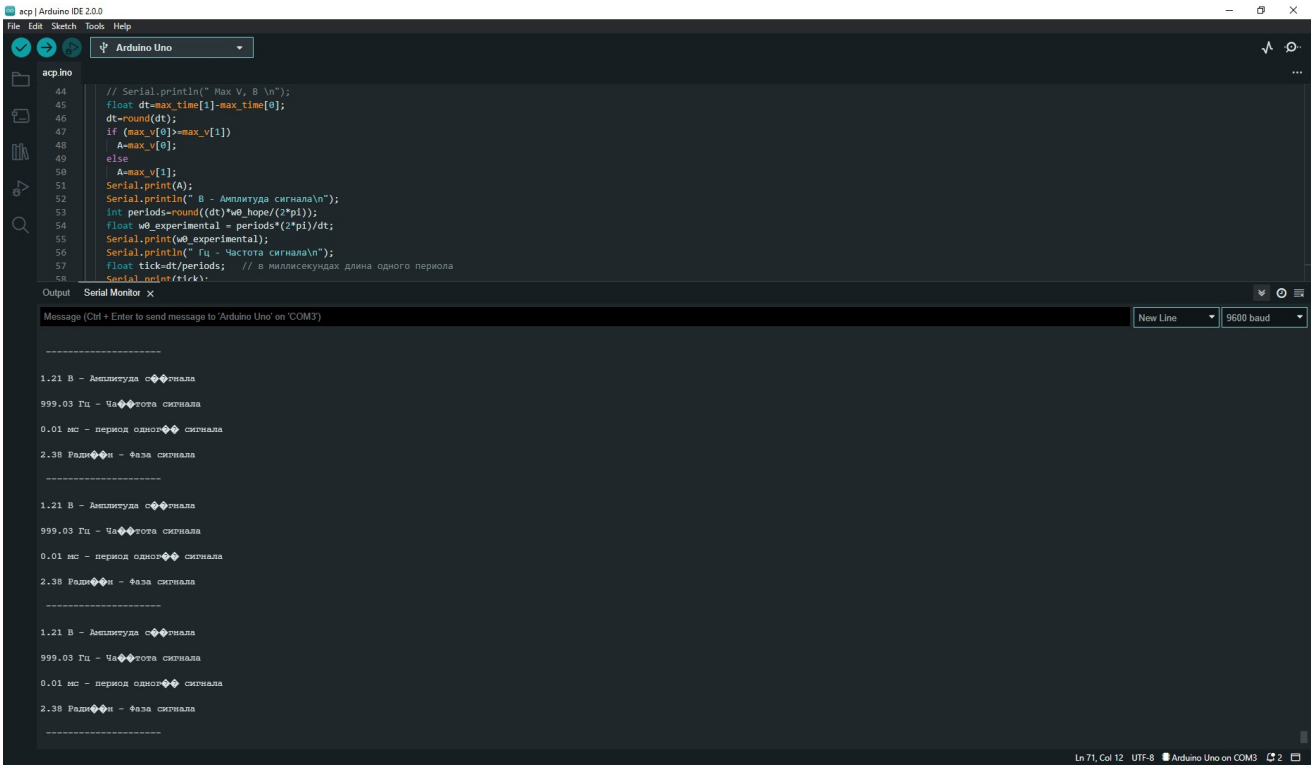


Реализация на ардуинке

Дальше уже переходим в Arduino IDE, где реализуем то же самое, но уже на МК. **Обратите внимание,** что пока прога была отлажена на пустом, фоновом сигнале, шуме, подаваемом на АЦП A0. Требуется дальнейшая проверка.

```
// the setup function runs once when you press reset or power the board
void setup() {
  Serial.begin(9600);

}

// the loop function runs over and over again forever
void loop() {
  int i = analogRead(A0);
  float A;
  static int num=0; // для циклов
  static int curr=0; // для запоминания номера сигнала
  static float max_time[2]={0,0};
  static float pi=3.14159265;
  num = num + 1;
  curr = curr + 1;
  static float max_v[2]={0,0};
  static int max_num[2];
  float w0_hope=1000;
  float w0_experimental=w0_hope;
  float v = 5.0 * i / 1023.0 ;// Преобразовываем уровень i  в напряжение
  // Serial.print(v);  // выводим напряжение
  // Serial.println("В.\n");  // Выводим доп текст
  // Serial.print(num);  // выводим
  // Serial.println(" - number \n");
  int max=1000;
  if (v>max_v[0])  // Запоминаем 2 максимальных значения в интервале max
  {
    max_v[0]=max_v[1];
    max_num[0]=max_num[1];
    max_time[0]=max_time[1];
    max_time[1]=millis();

    max_v[1]=v;
    max_num[1]=curr;

  }
  if (num==max) // Экспорт на порт компа значения максимальных значений, и их
```

```
номера
  {
    // Serial.print(max_v[0]);
    // Serial.println(" ");
    // Serial.print(max_v[1]);  // выводим
    // Serial.println(" Max V, В \n");
    float dt=max_time[1]-max_time[0];
    dt=round(dt);
    if (max_v[0]>=max_v[1])
      A=max_v[0];
    else
      A=max_v[1];
    Serial.print(A);
    Serial.println(" В - Амплитуда сигнала\n");
    int periods=round((dt)*w0_hope/(2*pi));
    float w0_experimental = periods*(2*pi)/dt;
    Serial.print(w0_experimental);
    Serial.println(" Гц - Частота сигнала\n");
    float tick=dt/periods;   // в миллисекундах длина одного периола
    Serial.print(tick);
    Serial.println(" мс - период одного сигнала\n");
    max_time[1]=max_time[1]-tick*floor(max_time[1]/tick);
    float phi=2*pi-w0_experimental*max_time[1];
    Serial.print(phi);
    Serial.println(" Радиан - Фаза сигнала\n");
    Serial.println(" --------------------\n");
    num=0;
    max_v[0]=0;
    max_v[1]=0;
    max_time[0]=0;
    max_time[1]=0;
  }
  delay(1);
  //delay(10);                      // wait for a second
  // Serial.print(millis());
  // Serial.println(" - Time\n");
}
```



вот что ардуинка передает компу

Я еще не реализовал вывод списка в отдельный файл, но постараюсь скоро и это сделать.

Также хочется сказать, что реализация подсчета фазы оставляет желать лучшего, лучше было бы после нахождение времени одного периода находить количество не прошедших на данных момент периодов для подсчета фазы, а запомнить начальные несколько значений, чтобы со временем числа не увеличивались. Жду замечаний, так как метод сырой.