МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет»

РТУ МИРЭА

Филиал РТУ МИРЭА в г. Фрязино
Кафедра «Конструирование и технология электронных средств»

Курсовая работа
по дисциплине
«Цифровая обработка сигналов»
на тему
«Обработка реального сигнала»

Выполнил студент группы ФКМО-01-20 Налогин И.А.

Принял преподаватель Сольвьев Н.А.

Реферат выполнен «___»_____2021 г.

«Зачтено»
«___»_____2021 г.

Фрязино 2021

Содержание

Цель работы

Необходимо реализовать многоканальную обработку сигнала, представленного в виде отсчётов.

Сигнал представлен в виде М-последовательности закодированной с помощью одноимпульсного фазокодоманипулированного сигнала. Структурно сигнал переставляет собой сумму двух компонент: условно сигнала передатчика, который не имеет доплеровского сдвига частоты, и отражённого сигнала со значительным доплеровским сдвигом.

- Несущая частота сигнала 10,2 GHz.
- Данные представлены в бинарном файле, тип данных float, размер 4 байта.
- Частота дискретизации — 12 MHz
- Период повторения — 180 usec
- Степень полинома для ФКМ — 5
- Скважность импульса — 5
- Частота ПЧ есть ¼ от частоты дискретизации.
- Базовое значение регистра сдвига [1, 0, 0, 0, 0]

Необходимо:

1. Произвести обработку сигнала наиболее оптимальным (по времени) методом.

2. Выделить отражённый сигнал и определить его параметры.

Введение

Рис 1: Радиолокационная станция

Цифровая обработка сигналов – это метод обработки информации, основанный на определенной последовательности с установленным периодом дискретизации.

Цифровыми называют сигналы, представленные с учетом уровневого и временного квантования. Такие сигналы широко применяются в самых разных отраслях и видах современной техники.

Основой для предварительной обработки сигналов являются процедуры быстрых дискретных ортогональных преобразований, которые реализуются в различных функциональных базисах, процедуры линейной и нелинейной фильтрации, линейной алгебры.

Самая главная задача обработки – это устранение помех и шумов. Решить эту задачу в полной мере можно только в том случае, если сигнал поступает избыточный с точно определенными параметрами.

В результате необходимо обеспечить правильный прием сигнала. Чем больше поступает полезного сигнала и чем меньше помех, тем больше вероятность выполнить задачу качественно.

К основным задачам цифровой обработке сигнала относится:

- спектральный анализ
- линейная фильтрация
- свертка традиционных типов
- частотно-временной анализ
- нелинейная обработка
- адаптивная фильтрация
- многоскоростная обработка
- секционная свертка

Цифровая обработка используется в таких отраслях, как:

- самолетостроение, оборонные системы, космическое оборудование
- электроника для автомобилей
- осветительные системы
- мобильная, стационарная связь, интернет-телефония и прочее
- электронные приборы и устройства для дома
- медицинское оборудование
- измерительные и прочие приборы
- системы управления
- средства для обеспечения безопасности

Радиолокация

Рис 2: Доплеровкий метеорологический радиолокатор

Радиолокация - метод обнаружения и определения местонахождения объектов посредством радиоволн. Эти волны излучаются радиолокационной станцией, отражаются от объекта и возвращаются на станцию, которая анализирует их, чтобы точно определить место, где находится объект.

Как и любое направление развития науки и техники, радиолокация базируется на некоторых физических основах, позволяющих обеспечивать решение стоящих перед ней задач, а именно: обнаруживать различного рода объекты и определять координаты и параметры их движения с помощью радиоволн.

Использование радиоволн, или, другими словами, электромагнитных колебаний (ЭМК), частотный диапазон которых сосредоточен в пределах от 3 кГц до 300 ГГц, определяет основные преимущества радиолокационных систем (РЛС) перед другими системами локации (оптическими, инфракрасными, ультразвуковыми). В первую очередь, это обусловлено тем, что закономерности распространения радиоволн в однородной среде достаточно стабильны как в любое время суток, так и в любое время года и, следовательно, изменение условий оптической видимости, обусловленных появлением дождя, снега, тумана или изменением времени суток, не нарушает работоспособность РЛС.

Основными закономерностями распространения радиоволн, которые позволяют обнаруживать объекты и измерять координаты и параметры их

движения, являются следующие:

- постоянство скорости и прямолинейность распространения радиоволн в однородной среде (при проведении инженерных расчетов скорость распространения радиоволн принимают равной $3 \cdot 10^{-8}$ м/с
- способность радиоволн отражаться от различных областей пространства, электрические или магнитные параметры которых отличаются от аналогичных параметров среды распространения
- изменение частоты принимаемого сигнала по отношению к частоте излученного сигнала при относительном движении источника излучения и приемника радиолокационного сигнала (эффект Доплера)

М-последовательность

М-последовательность(последовательность максимальной длины) — псевдослучайная двоичная последовательность, порожденная регистром сдвига с линейной обратной связью и имеющая максимальный период. М-последовательности применяются в широкополосных системах связи.

М-последовательности обладают следующими свойствами:

- М-последовательности являются периодическими с периодом $N = 2^N - 1$
- количество символов, принимающих значение единица, на длине одного периода М-последовательности на единицу больше, чем количество символов, принимающих значение нуль
- любые комбинации символов длины на длине одного периода М-последовательности за исключением комбинации из нулей встречаются не более одного раза. Комбинация из нулей является запрещённой: на её основе может генерироваться только последовательность из одних нулей
- сумма по модулю 2 любой М-последовательности с её произвольным циклическим сдвигом также является М-последовательностью
- периодическая АКФ любой М-последовательности имеет постоянный уровень боковых лепестков, равный (-1/N)
- АКФ усечённой М-последовательности, под которой понимается непериодическая последовательность длиной в период N, имеет величину боковых лепестков, близкую к .. Поэтому с ростом N величина боковых пиков уменьшается.

Фазокодовая манипуляция

Фазокодовая манипуляция это такая модуляция зондирующего сигнала, при поторой устанавливается определенная последовательность изменений начальной фазы высокочастотного колебания в кодовых интервалах, на которые непрерывно разбивается весь период передачи.

Любой сигнала может быть рассмотрен как сумма сдвинутых друг относительно друга на определенную величину элементарных последовательностей прямоугольных импульсов, каждая из которых образована колебаниями в кодовых интервалах, одинаково расположенных в соседних периодах модуляции.

Основными характеристиками кодов являются длительность кодового интервала — она определяет разрешающую способность по дальности и количество кодовых интервалов — которые определяют количество дополнительных максимумов, которые мешают при раскодировке.

Ход работы:

Генерация m-последовательности

Для генерации последовательности максимальной длины хорошо подходит функция max_len_seq из библиотеки scipy.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.max_len_seq.html

pnseq = signal.max_len_seq(nbits=PN_ORDER, state=PN_STATE, length=PN_LENGTH, taps=PN_POLYNOM)[0]

где:

- PN_ORDER — порядок полинома — по условию 5
- PN_STATE - начальное состояние сдвигового регистра в условии не задано, примем его равным [1, 0, 0, 0, 0]
- PN_LENGTH — длина выходного полинома, для максимальной длины — $2^{PN\_OREDER} — 1 = 31$
- PN_POLYNOM — полином для генерации — [5, 3, 0] по условию

На выходе мы получаем последовательность максимальной длины и итоговое состояние сдвигового регистра — нам интересна только последовательность, поэтому берем только первый элемент результата.

Результат - **[1 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0]**

В целом в таком виде нам использовать ее неудобно, так как при модуляции у нас при наличии сигнала будут смещение частоты вверх и вниз, соответственно преобразуем все 0 в -1 домножив каждый элемент результата на 2 и отняв от него 1.

pnseq = (pnseq << 1) — 1

*умножение на 2 заменено битовым сдвигом влево, для целых чисел это эквивалентные операции.

Результат - **[1 -1 -1 -1 -1 1 -1 1 -1 1 1 1 -1 1 1 -1 -1 -1 1 1 1 1 1 -1**

-1 1 1 -1 1 -1 -1]

Генерация опорного сигнала:

Опорный сигнал может быть сделан частотно или амплитудно:

- **Амплитудно** выполняется с помощью метода ближайшего соседа, в результате получается точное значение амплитуды, но неточное значение фазы
- **Частотно** сигнал представляется с помощью суммы спектров прямоугольных импульсов, смещенных относительно начала координат, в результате получается точное значение фазы, но неточное значение амплитуды

Для каждого из методов нужно выбрать пониженную достаточную частоту дискретизации, которая выбирается исходя из теоремы Котельникова — в нашем случае наименьшей длительностью обладает 1 импульс из которых мы строим опорный сигнал, соответственно период пониженной достаточной частоты дискретизации должен быть как минимум вдвое ниже длины этого импульса.

Для хорошего представления эта цифра должны быть не менее 4, выберем 4. Число отсчетов для удобной обработки должно быть также кратно 4.

TAU = (PERIOD / (DUTY * PN_LENGTH))

samp_lo_time = TAU/scale

samplecount = round(PERIOD/samp_lo_time)&(~0x03)

Побитовое «И» с числом у которого все биты кроме нижних трех равны 1 делает так, что число нацело делится на 4.

Пониженная достаточная частота дискретизации **34MHz**

Количество отсчетов в опорном сигнале **620**

Листинг функции, которая рассчитывает необходимую пониженную частоту и количесво отсчетов в опорном сигнале.

```
"""
calculate sampledata parameters
@param scale        - scale factor (should be 4 ... 6)

@return             - samplecount and sample freq
"""
def signaldata_get(scale):
    global SAMPLE_FREQ
    global PERIOD
    global PN_LENGTH
    global TAU

    if(4 > scale or 6 < scale):
        print("low freq mult is ", scale,"this is out of range 4 .. 6, low freq
mult will be set to 4")
        scale = 4

    #period in samples
    #sample_time = 1/SAMPLE_FREQ
    #period_s = round(PERIOD/sample_time)
    #print("period contain ", period_s, " samples")

    #set lowered discretization time and freq
    samp_lo_time = TAU/scale
    #count of samples with lowered freq
    samplecount = round(PERIOD/samp_lo_time)&(~0x03)
    #clarify values after samplecount calculation
    samp_lo_freq = samplecount/PERIOD
    samp_lo_time = 1/samp_lo_freq

    print("lowered period : ", '%.f4' %(samp_lo_time*1000000), " usec")
    print("lowered freq :   ", '%.f4' %(samp_lo_freq/1000000), " MHz")
    print("samplecount :    ", samplecount)

    return (samplecount, samp_lo_freq)
```

Амплитудно:

В ходе генерации опорного сигнала выясняется, к какому участку принадлежит тот или иной отсчет — к участку в котором значение сигнала -1, 1 или 0 и ему присваивается соответствующее значение.

На выходе функции получаем сигнал, затем его спектр с помощью прямого преобразования Фурье

Получившейся сигнал

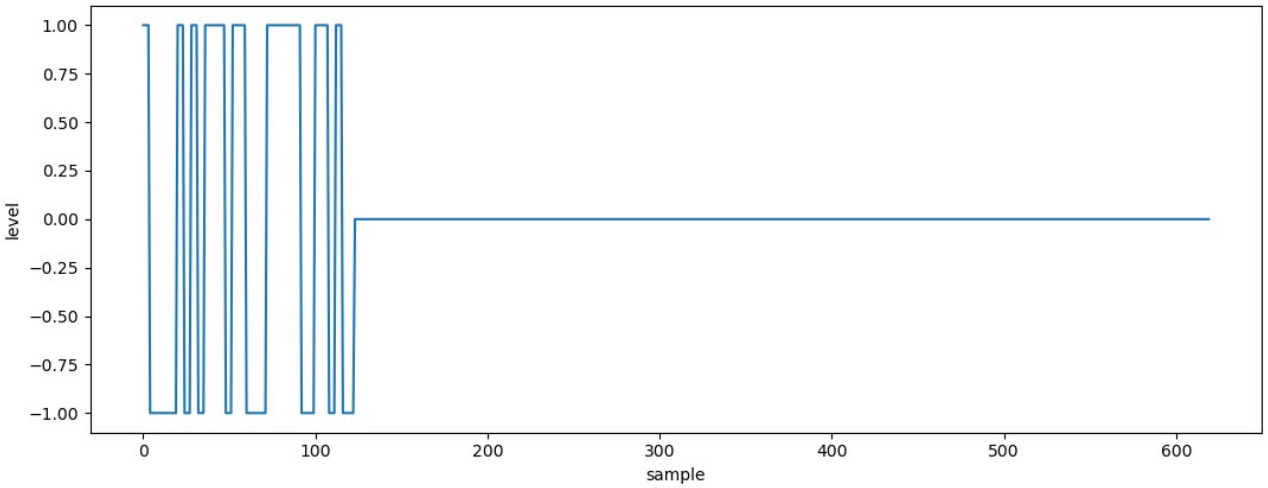


Рис 3: Амплитудно сгенерированный сигнал

Листинг функции генерации опорного сигнала амплитудно

```
"""
solve na_2021 by nearest neighbor method
@param pnseq        - pn sequence
@param samplecount  - estimated number of samples

@return             - signal and spectre of refernece signal
"""
def nearest_neighbor(pnseq, samplecount):
    global SAMPLE_FREQ
    global PERIOD
    global PN_LENGTH
    global TAU

    samp_lo_freq = samplecount/PERIOD

    #calc last boundary in sample
    lastnum = math.floor(PN_LENGTH*samp_lo_freq*TAU)

    nn_signal_s = np.zeros(samplecount)

    index_translator = 1/(samp_lo_freq*TAU)
    for i in range(lastnum):
        sel = math.floor(i*index_translator)
        nn_signal_s[i] = pnseq[sel]

    if(2 == NA_DEBUG):
        plt.xlabel("sample")
        plt.ylabel("level")
        plt.plot(nn_signal_s)
        plt.show()

    nn_spectre = np.fft.fft(nn_signal_s)
    return (nn_signal_s.reshape(samplecount), nn_spectre.reshape(samplecount))
```

Частотно:

Спектр сигнала представляется в виде суммы спектров прямоугольных имульсов с амплитудой 1 или -1, смещенным относительно начала координат. Для получения сигнала проводится обратное преобразование Фурье.

Спектр прямоугольного импульса, с началом в начаеле координат и смещенного относительно начала координат на pulseoffset:

$$S_{nu}(\omega_n) = A * sinc(\frac{\omega_n t}{2}) * \exp(-i\,\omega_n(t/2 + pulseoffset))$$

Соответственно, для получения итогового спекрта суммируется каждый из PN_LENGTH импульсов.

Получившейся сигнал:

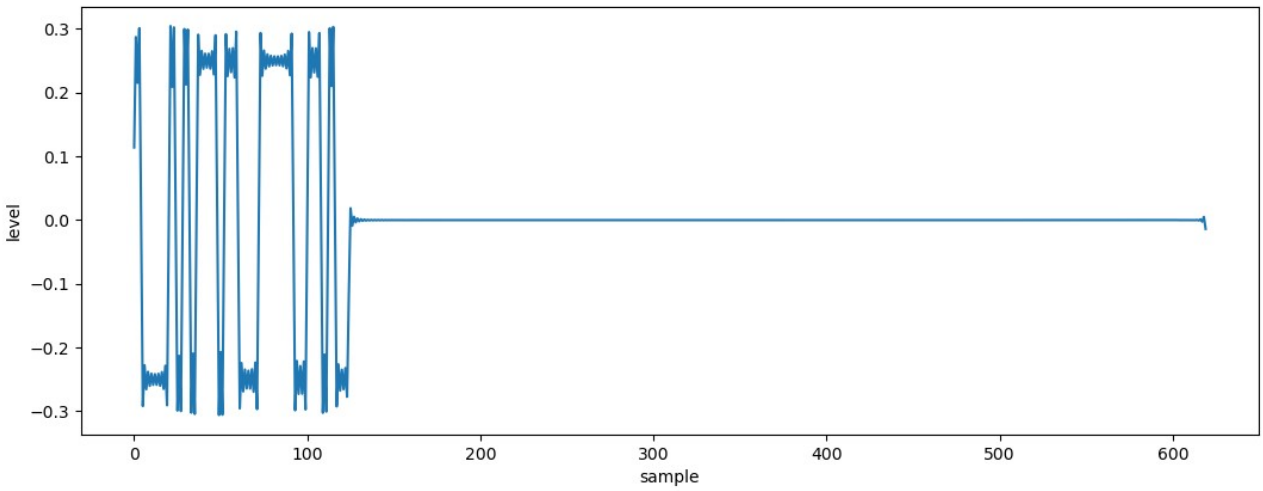


Рис 4: Частотно сгенерированный сигнал

В работе будет применяться опорный сигнал, полученный частатно.

Листинг функции генерации опорного сигнала частотно

```
"""
solve na_2021 by nearest neighbor method
@param pnseq        - pn sequence
@param samplecount  - estimated number of samples

@return             - signal and spectre of reference signal
"""
def freq_based(pnseq, samplecount):
    global SAMPLE_FREQ
    global PERIOD
    global PN_LENGTH
    global TAU

    fb_signal_s = np.empty(samplecount)
    mid = samplecount>>1

    fb_signal_s[:mid] = np.arange(0,mid)
    fb_signal_s[mid:] = np.arange(-mid,0)
    #optimize fb_signal = (fb_signal * samp_lo_freq)/samplecount,
    #where samp_lo_freq is samplecount/period
    fb_signal_s /= PERIOD

    if(3 == NA_DEBUG):
        plt.xlabel("sample")
        plt.ylabel("level")
        plt.plot(fb_signal_s)
        plt.show()

    #reshape signal to simlify arith
    fb_signal_s = fb_signal_s.reshape(1,samplecount)
    #reshape pnseq to simplify arith
    pnseq = pnseq.reshape(1,PN_LENGTH)
    #pulses moved on 0.5 to make first pulse front on start of coordinates
    timeline = (np.arange(0, PN_LENGTH) + 0.5)*TAU
    #reshape timeline
    timeline = timeline.reshape(PN_LENGTH,1)

    timefreq = np.dot(timeline, fb_signal_s)
    delay_part = np.exp(-2j*np.pi*timefreq)
    delay_part = np.dot(pnseq, delay_part)
```

```
signal_part = np.sinc(fb_signal_s*TAU)

if(4 == NA_DEBUG):
    print("shape of signal part : ", signal_part.shape)
    print("shape of delay part  : ", delay_part.shape)

fb_spectre = signal_part*delay_part

fb_signal = np.fft.ifft(fb_spectre)
if(4 == NA_DEBUG):
    plt.xlabel("sample")
    plt.ylabel("level")
    plt.plot(np.real(fb_signal.reshape(samplecount,1)))
    plt.show()

return (fb_signal.reshape(samplecount), fb_spectre.reshape(samplecount))
```

Получение исходных данных из файла и представление их в удобном для обработки виде

Данные представлены в сыром виде в файле, откуда их необходимо прочитать в переменную для дальнейшей работы:

na_data = np.fromfile(filename, dtype=np.single)

Для удобной обработки сигнала необходимо представить исходные данные в виде матрицы, каждая строка которой имеет длину в период опорного сигнала.

sample_time = 1/SAMPLE_FREQ

period_s = round(PERIOD/sample_time)

Неполные периоды, если они присуствуют, необходимо отбросить или дополнить нулями.

Листинг подготовки исходных данных к обработке

```
#slice data by samplecount chunks for further processing
na_data_len = na_data.size
na_pr_count = round(na_data_len/period_s)

#check if we can't make roudly from data matrix (period_s, length/period_s)
if(na_pr_count*period_s != na_data_len):
    #data have tail - cut off tail
    samples_dropped = na_data_len - na_pr_count*period_s
    print("data have tail. ", samples_dropped, " samples was dropped")
    na_data = na_data[:na_pr_count*period_s]
else:
    #data have not tail
    print("data have not tail")

#length of all input signal, used when we will calculate the position
total_time = (na_pr_count*period_s)/SAMPLE_FREQ

#reshape input data
na_data = na_data.reshape(na_pr_count,period_s)
```

Исследование сигнала

Для получения информации о расстоянии необходимо знать время излучения импульса и время приема отраженного импульса.

Исследование наличия сигнала выполняется с помощью сжатия импульса опорного сигнала и исследуемого периода.

Фазы опорного сигнала и сигнала в исходных данных не одинаковы и для корректировки фазы умножение должно быть произведено на комплексно сопряженный спектр опорного сигнала.

Листинг сжатия импульса

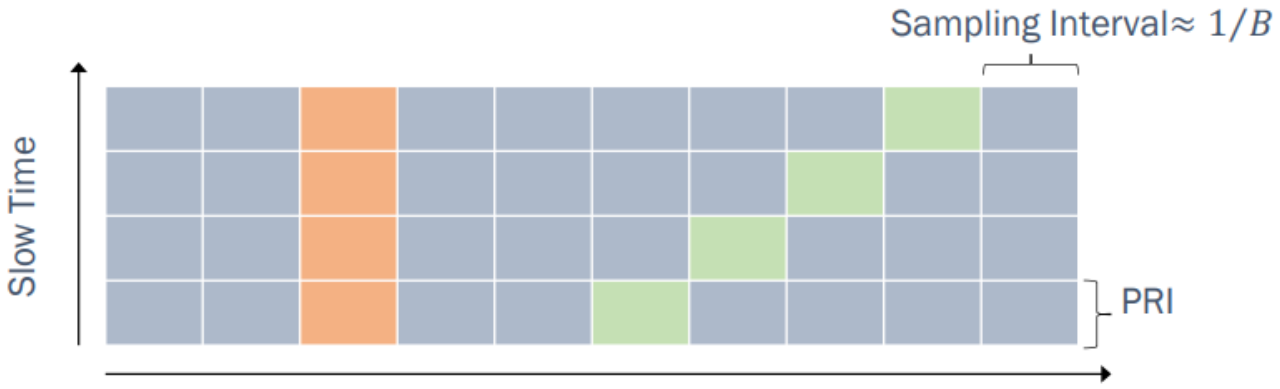
```
"""
make pulse compression
@param spectre_ref      - reference sigal spectre
@param spectre_signal   - signal spectre

@return                 - impulse
"""
def pulse_compression(spectre_ref, spectre_signal):
    cv_ref = np.conj(spectre_ref)
    cp_spectre = cv_ref*spectre_signal
    cp_pulse = np.real(np.fft.ifft(cp_spectre))

    if(7 == NA_DEBUG):
        plt.suptitle("compressed impulse")
        plt.xlabel("sample")
        plt.ylabel("level")
        plt.plot(abs(cp_pulse))
        plt.show()


    return cp_pulse
```

В силу того, что импульс довольно слабый, для его детектирования необходимо произвести обработку в медленно времени с когерентным накоплением - выборку откликов каждого излучаемого импульса с интервалом, равным разрешению радиолокатора по дальности, и просуммировать отклики от NA_PR_COUNT импульсов. После накопления суммы NA_PR_COUNT импульсов выполняется амплитудное детектирование.



Результат когерентного накопления отправленного импульса

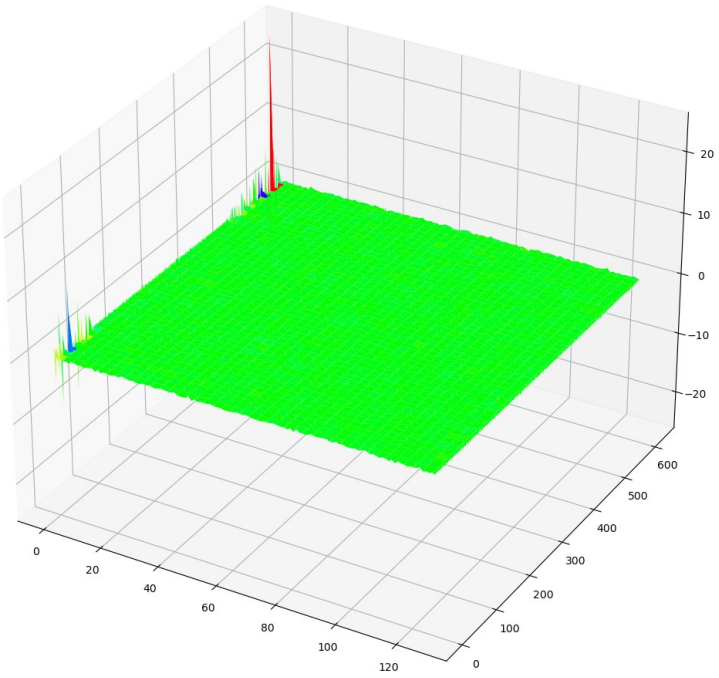


Рис 5: Видим пичок когерентного накопления

Листинг когерентного накопления

```
"""
make coherent accumulation of input data,
@param na_data          - input samples, sliced by period into matrix
@param pos              - pos for spectre moving(in every line)
@param ref_spectre      - spectre of reference signal

@return                 - accumulated data
"""
def coherent_accumulation(na_data, pos, ref_spectre):
    #process in slow-time the input data
    na_pr_count = na_data.shape[0]
    samplecount = ref_spectre.shape[0]

    na_proc_array = np.empty((na_pr_count, samplecount))
    for i in range(na_pr_count):
        na_data_period = na_data[i]
        line_spectre = move_spectre(na_data_period, pos, samplecount)
        pulse = pulse_compression(ref_spectre, line_spectre)
        na_proc_array[i] = pulse

    #reshape matrix and make fft in every line, then reshape back (for coherent
burst accumulation)
    coh_acc_na_array = na_proc_array.T
    for i in range(samplecount):
        coh_acc_na_array[i] = np.fft.fft(coh_acc_na_array[i])

    coh_acc_na_array = coh_acc_na_array.T
    coh_acc_na_array = np.abs(coh_acc_na_array)

    return coh_acc_na_array
```

Отраженный сигнал:

Из условия отраженный сигнал обладает значительным доплеровским сдвигом, соответственно прямое сравнение его спектра со спектром опорного сигнала невозможно.

Для получения информации об отраженном сигнала уберем из спектра все, что связано с сигналом отправленным и будем проводить сравнение со спектром опорного сигнала, сдвинутым циклически, чтобы учесть доплеровский сдвиг частоты опорного сигнала.

Удаление из спектра будем проводить занулением соответствующих отправленному сигналу линий матрицы обработки в медленном времени

cacc_data[range(10)] = 0

В силу того, что заранее неизвестно, где находится сигнал введем критерий наличия сигнала — примем за критерий наличия сигнала пик с высотой, превышающей уровень шума на графике, высота которого максимально и количество пиков, с высотой до половины не более 4 единиц.

lmax = np.amax(coh_acc_data)

threshold_mid = lmax/2

cmax = (coh_acc_data > threshold_mid).sum()

Листинг определения наличия сигнала

```
"""
check, that signal present in data
@param coh_acc_data     - coherent accumulated data
@param threshold        - signal thrshold

@return                 - signal coordinates, if present or null, if not
"""
def signal_is_present(coh_acc_data, threshold, dbg_num):

    #number of samples, that more then threashold should be 1
    lmax = np.amax(coh_acc_data)

    if(threshold > lmax):
        #we have not peak
        return None

    #we should not have lot of peaks with half of lmax amplitude
    threshold_mid = lmax/2
    cmax = (coh_acc_data > threshold_mid).sum()
    if(4 < cmax):
        #we have too wide spectrum
        return None

    #we have peak - return it's value and index
    if(11 == NA_DEBUG):
        maxvalues = np.amax(coh_acc_data, axis=0)
           suptitle = "offset num : " + str(debug_num) + "\npulses more then
threshold : " + str(cmax) + "\nmaxvalue : " + str(lmax)
        plt.suptitle(suptitle)
        plt.xlabel("sample")
        plt.ylabel("level")
        plt.plot(abs(maxvalues))
        plt.show()

    if(12 == NA_DEBUG):
        plot_array = coh_acc_data
        #print surface plot for debugging
            suptitle = "offset num : " + str(dbg_num) + "\npulses more then
threshold : " + str(cmax) + "\nmaxvalue : " + str(lmax)
        fig = plt.figure()
```

```
        plt.suptitle(suptitle)
        surf3d = fig.add_subplot(111, projection='3d')
        X = np.arange(0, na_pr_count, 1)
        Y = np.arange(0, samplecount, 1)
        X, Y = np.meshgrid(X,Y)
        Z = plot_array[X,Y]

        #print("shape of Z is ", Z.shape)

        surf3d.plot_surface(X, Y, Z, cmap=plt.cm.get_cmap('hsv'))
        plt.show()

           na_argmax   =   np.unravel_index(np.argmax(coh_acc_data,   axis=None),
coh_acc_data.shape)
    return (lmax, na_argmax[1], na_argmax[0])
```

Поиск по различным смещениям спектра опорного сигнала

После того, как был определен критерий наличия сигнала остается только проитись по возможным смещениям частоты опорного сигнала для определения того, есть ли, и если есть — то на какой частоте и когда приходит отраженный сигнал.

moved_ref_spectre = np.roll(ref_spectre, freqshift)

cacc_data = coherent_accumulation(na_data, pos, moved_ref_spectre)

Мощность отраженного сигнала невысока и для определения того, какой из отраженных сигналов(которые появились во многом из-за дискретных шагов обработки и боковых лепестков отраженного импульса) будет взят в обработку предположим, что правильным является тот сигнал, мощность которого максимальна.

В ходе работы было выяснено, что отраженный сигнал находится при сдвиге в 22 отсчета.

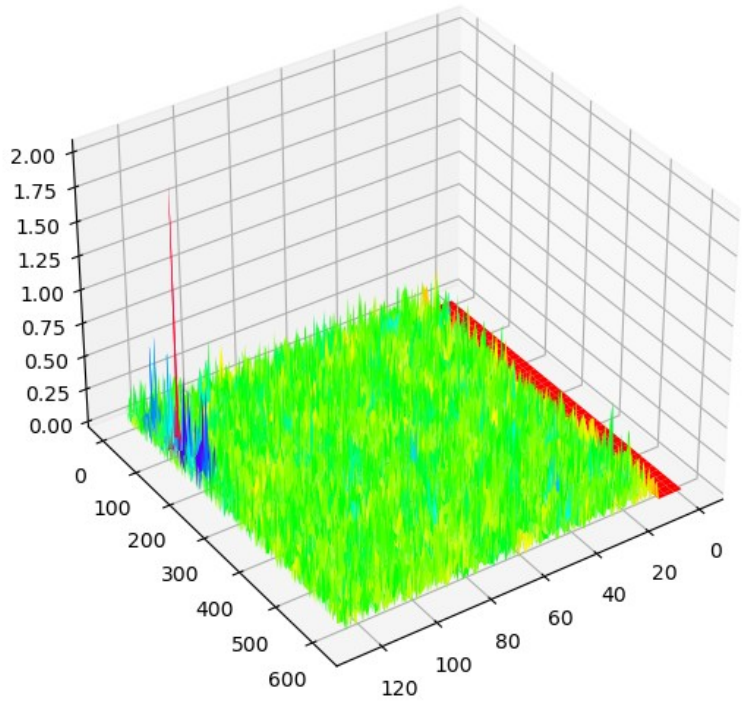


Рис 6: Видим зануленный отправленный сигнал и пичок отраженного сигнала

Листинг поиска отраженного сигнала

```
 #get point of input signal
    starttime = time.time()
    fp_data = coherent_accumulation(na_data, pos, ref_spectre)
    founded = signal_is_present(fp_data, CH_THRESHOLD, 0)
    end_time = round((time.time() - starttime)*1000000)

    if founded is None:
        print("tx peak is not founded, signal was not transmitted")
        sys.exit(os.EX_SOFTWARE)

    fp_peak_max = founded[0]
    fp_peak_pos = founded[1]
    print("first peak max is ", '%.3f' %fp_peak_max, " located at ", founded[1],
" sample\none freq processing take ", end_time, " useconds")



    #we shifte the spectre right and left by samplecount/8
    shift_max = samplecount >> 3

    starttime = time.time()
    #used to find the value with max peak
    max_pos = 0
    max_value = 0
    max_freqshift = 0
    max_freq = 0

    #step one - spectre is shifted to right
    for freqshift in range (-shift_max, +shift_max):
        moved_ref_spectre = np.roll(ref_spectre, freqshift)

        cacc_data = coherent_accumulation(na_data, pos, moved_ref_spectre)

        #cutoff transmitted signal (it's send at zero time and it's petals
        cacc_data[range(10)] = 0

        founded = signal_is_present(cacc_data, CH_THRESHOLD, freqshift)
        if founded is not None :
            peak_max = founded[0]
            peak_pos = founded[1]
```

```
            peak_freq = founded[2]
            if(13 == NA_DEBUG):
                #don't print peak info, if we meashuring the time of
calculations
                print("peak found.\n\tvalue : ", '%.4f' %peak_max, "\n\tpos : ",
peak_pos, "\n\tfreqshift : ", freqshift)
            if(max_value < peak_max):
                max_value = peak_max
                max_pos = peak_pos
                max_freqshift = freqshift
                max_freq = peak_freq

    time_passed = round((time.time() - starttime)*1000000)
```

Получение информации о сигнале

На данном этапе у нас есть время отправки сигнала, время получения сигнала и сдвиг частоты в отсчетах принятого сигнала. Зная скорость распространения сигнала в среде и частоты несущей можно определить параметры сигнала.

Из-за особенностей обработки в медленном времени может оказаться так, что принятый сигнал находится ближе к началу координат, чем отправленный — в таком случае к разнице во времени между принятым и отправленным сигналом нужно добивить 1 период(периоды сигнала идут последовательно и такое расположение сигнала — чистая условность).

sample_delta = max_pos - fp_peak_pos

if(0 > sample_delta):

    sample_delta += samplecount

- Расстояние до цели — это половина произведения скорости распространения волны на время до цели (половина, так как волне нужно дойти до цели и возвратиться).
- Скорость цели — это половина произведения скорости волны в среде умножить на Допплеровскую частоту и разделить на частоту несущей(из основ радиолокации).

В результате было получено, что цель находится на расстоянии **6489 м** и приближается со скоростью **1799 м/с**.

Листинг получения информации о сигнале

```
if(0 < max_value):
         #our peaks is located reapetedly in timeline, so if tx peak located is after the rx peak,
        # than we need to add the period to delta (delta should not be negative)
        sample_delta = max_pos - fp_peak_pos
        if(0 > sample_delta):
            sample_delta += samplecount

        lowered_discretization_freq = (samplecount/period_s)*SAMPLE_FREQ
        time_delta = sample_delta/lowered_discretization_freq

         #signal move to target and back, so only first part of time should be takes, when we calculate the range
        target_range = WAVE_SPEED*time_delta/2

        freq_base = CARRIER_FREQ
        print("freq_base : ", freq_base)
        freq_doppler = (peak_freq/period_s + max_freqshift)/PERIOD
        print("freq_doppler : ", freq_doppler)
        #freq_doppler = (20 + 120/128)/PERIOD

        #calculate speed from doppler effect, using radar formula = dF = 2VF0/c ->
        target_speed = (WAVE_SPEED * freq_doppler) / (2 * freq_base)

        print("located signal present\n\tat_range: ", round(target_range))
        if((0.01) < target_speed):
            print("\n\tapproaching with speed: ", '%.3f'%target_speed)
        elif((-0.01) > target_speed):
            print("\n\tretreating with speed: ", '%.3f'%abs(target_speed))
        else:
            print("\n\tno target motion")
    else:
        print("target not present")
```

Общий вид результата работы скрипта:

Программа предназначена для запуска из консколи в возможностью перенаправления вывода не необходимое устройство. В данной версии настройки для академических целей являются глобальными переменными, однако в случае необходимости могут быть переданы как аргументы программы(для потокового использования на месте проведения исследований).

```
NA begin

Opening file DSP_02_FD30M.txt
file size =  2764800  B
contain  691200  4B single entries
generating pn-sequence, params:
      order 5
      polynom [5, 3, 0]
      start state [1, 0, 0, 0, 0]
      length 31

pn sequence:  [1 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0]
pn sequence:  [ 1 -1 -1 -1 -1  1 -1  1 -1  1  1  1 -1  1  1 -1 -1 -1  1  1  1  1
1 -1 -1  1  1 -1  1 -1 -1]
calculated tau, ms:  1.1613
period contain  5400  samples
lowered period :  04  usec
lowered freq :    34  MHz
samplecount :     620

method2: freq-based

data have not tail
reshaped na_data have shape  (128, 5400)
first peak max is  61.731  located at  596  sample
one freq processing take  36791  useconds
freq_base :  10200000000
freq_doppler :  122348.76543209875
located signal present
      at_range:  6489
      approaching with speed:  1799.247
calculation take : 5391502 useconds

NA success
```

Вывод:

В рамках этой курсовой работы были получены знания по определению параметров цели с помощью анализа зондирующего опорного сигнала радиолокатора и принятой радаром информации.

Для выполнения математических операций использовался язык Python и библиотека NumPy — это компоненты с открытым исходным кодом и поддержкой большинства платформ, используемых в прикладном программировании.

Работа была выполнена наиболее оптимальным по времени методом за счет выноса тяжелых математических операций за пределы интерпретатора в потоковую матричную обработку силами библиотеки NumPy.

В результате было определено положение — **6489 м** и скорость приближения цели — **1799 м/с.**

Все поставленные цели были достигнуты.

Git с этой работой, скриптом и последними правками можно скачать по адресу:

https://github.com/egan-ru/NA2021_cw

Список используемой литературы:

- Теоретические основы радиолокации : учебное пособие для студентов вузов / [Я.Д. Ширман, В.Н. Голиков, И.Н. Бусыгин и др.] ; под ред. Я.Д. Ширмана
- Цифровая обработка сигналов. Сергиенко А.Б. Питер, 2003г. ISBN: 5-318-00666-3
- Think DSP: Digital Signal Processing in Python, 2012 ISBN 13: 9781491938454
- Строганов А.В. Реализация алгоритмов цифровой обработки сигналов в базисе программируемых логических интегральных схем. Уч. пособие, 4-е изд., испр. и доп. 2019, изд. Лань