Оглавление

Александр Федотов

ПМиК 1 маг. ФИиИТ

1. Описание алгоритма

Для начала работы со звуковым файлом воспользуемся функцией `read()` из пакета `scipy`:

```
Fs, data = scipy.io.wavfile.read('kdt_410_8.wav')
```

`Fs` – частота дискретизации (для данного файла Fs = 8000).

`data` – наш сигнал в виде массива значений.

Сделаем взаимно однозначное отображение в отрезок [-1, 1]:

```
data = data / data.max()
```

И возьмем 0.2 секунды вокализованного фрагмента.

То есть Fs * 0.02 = 8000 * 0.02 = 160 отсчётов.

Выберем фрагмент, например с 6049 отсчёта и выведем график на экран:

```
N = Fs * 0.02
s = data[6049:6049 + N]
```

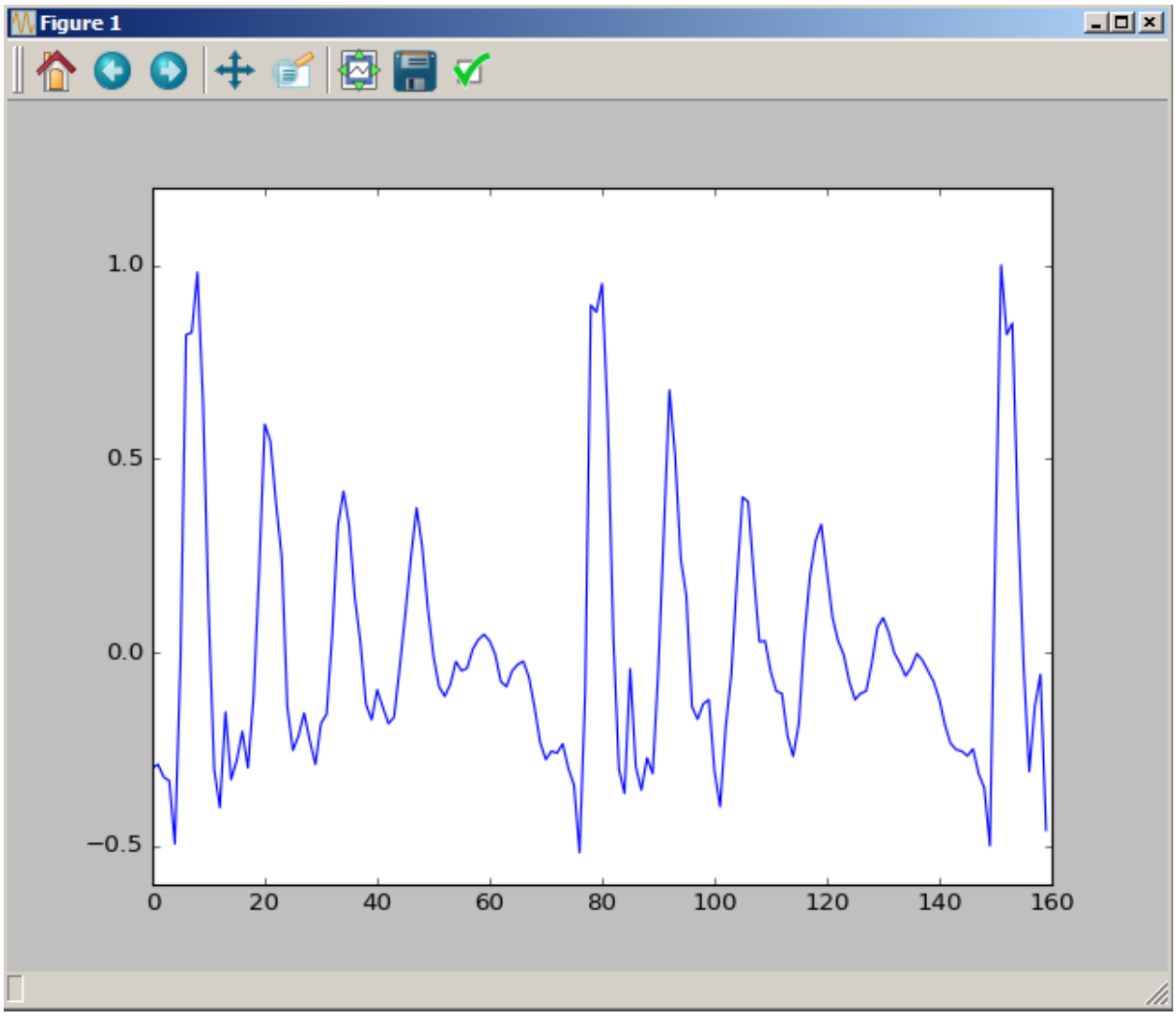

Рис. 1. Вокализованный фрагмент

Вычислим автокорреляционную функцию для полученного фрагмента. Возьмем значения `eta` в отрезке целых чисел от -10 до 10.

```
M = 10
r, eta = lpc.xcorr(s, M, 'biased')
```

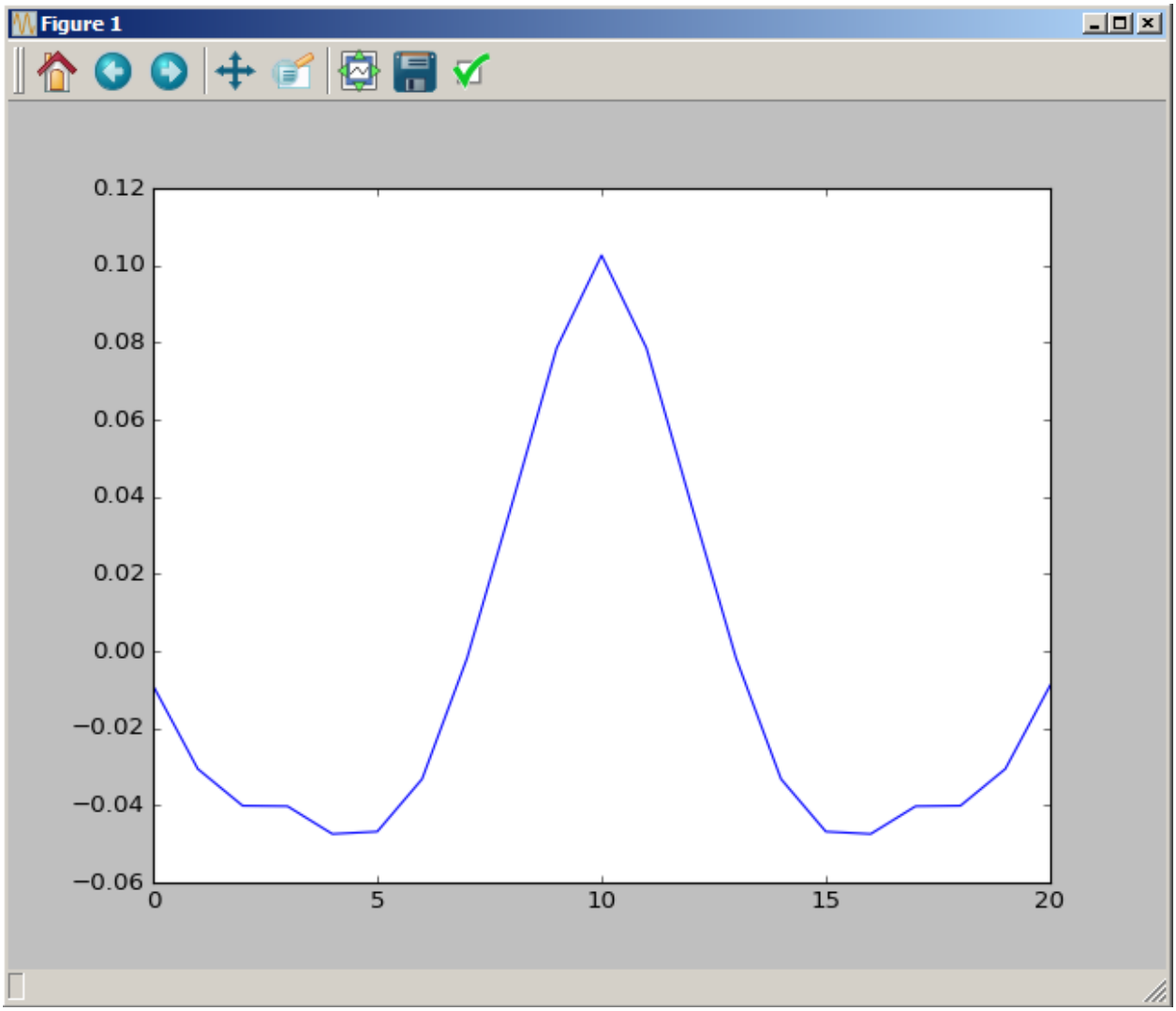

Рис. 2. Автокорреляционная функция

Теперь решим систему уравнений методом Левинсона-Дарбина, чтобы получить коэффициенты линейного предсказания и коэффициенты линейного отражения.

```
a, e, k = lpc.durbin(r[M:],M)
```

```
a= [ 1.12786067 -0.52931605  0.15143047 -0.43907405  0.26618171 -
0.26093165 0.54586437 -0.80841328  0.27985128  0.03085888]
e= [ 0.10260477  0.04243206  0.03081508  0.02997817  0.02893147
0.02892185 0.0281549   0.02775094  0.02028457  0.0182724   0.018255  ]
k= [ 0.76580115 -0.52323822 -0.16480075 -0.18685668  0.01823165 -
0.16284317-0.11978206 -0.51869974  0.31495572  0.03085888]
```

Рассмотрим как рассчитать частотный отклик.

Формируем сигнал с помощью окна Хэмминга и преобразования Фурье:

```
NFFT = 1024
X = np.abs( fft(s * hamming(N), NFFT) )
```

Получается следующий сигнал:

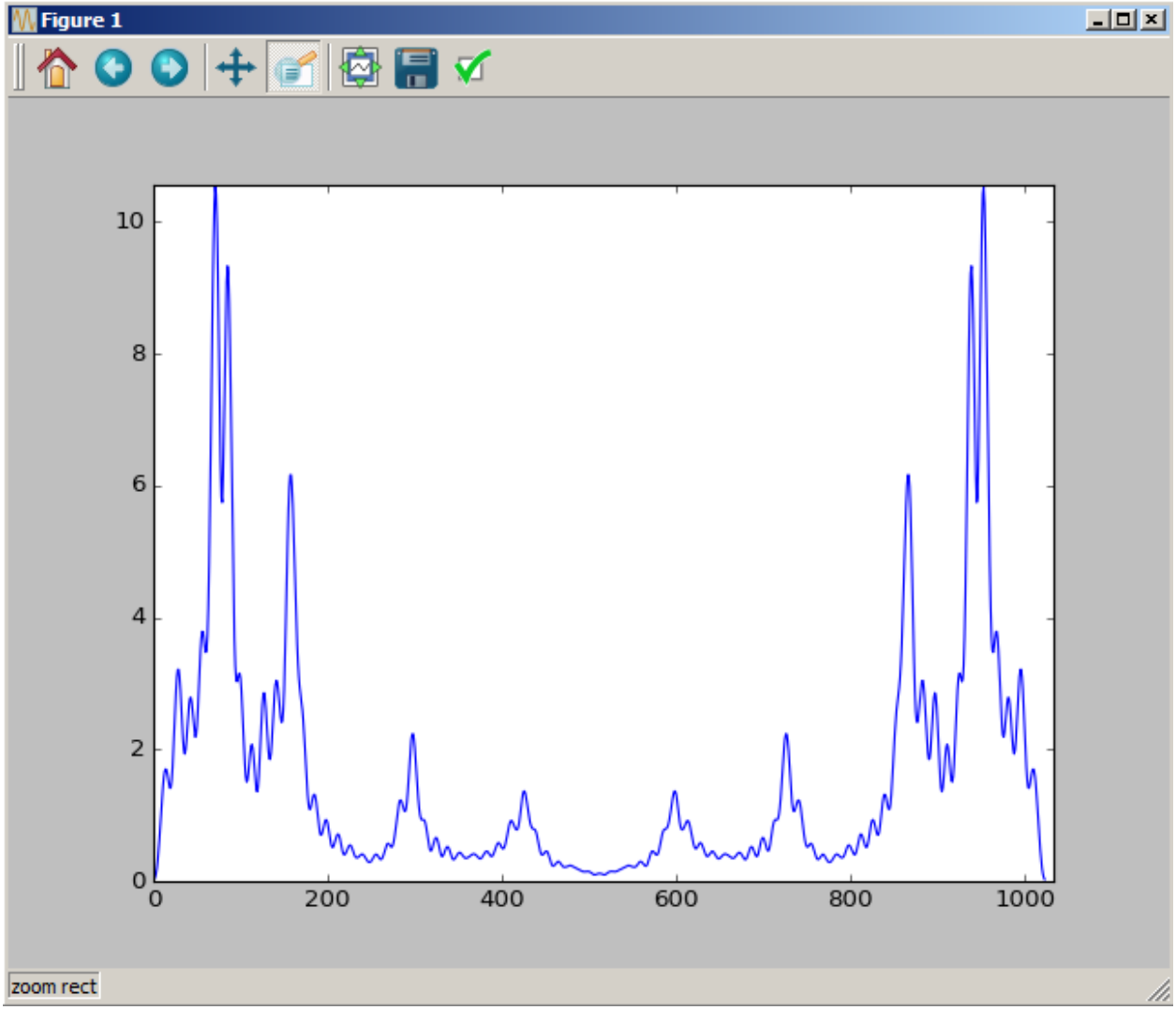

Рис. 3. Сигнал после преобразования Фурье

Создадим массив на 512 элементов:

```
k = np.arange(0, NFFT//2)
```

Делаем конкатенацию матрицы `[[1]]` с элементами матрицы `a`, взятых с обратным знаком:

```
denum = np.hstack( [np.matrix(1), -np.matrix(a)] )
Theta = 1 * np.abs( 1 / fft(denum, NFFT) )
```

Рисуем график:

```
fig,ax = plt.subplots()
ax.plot( 2*k / NFFT * Fs/2, 20 * np.log10(Theta[0,k]) )
```

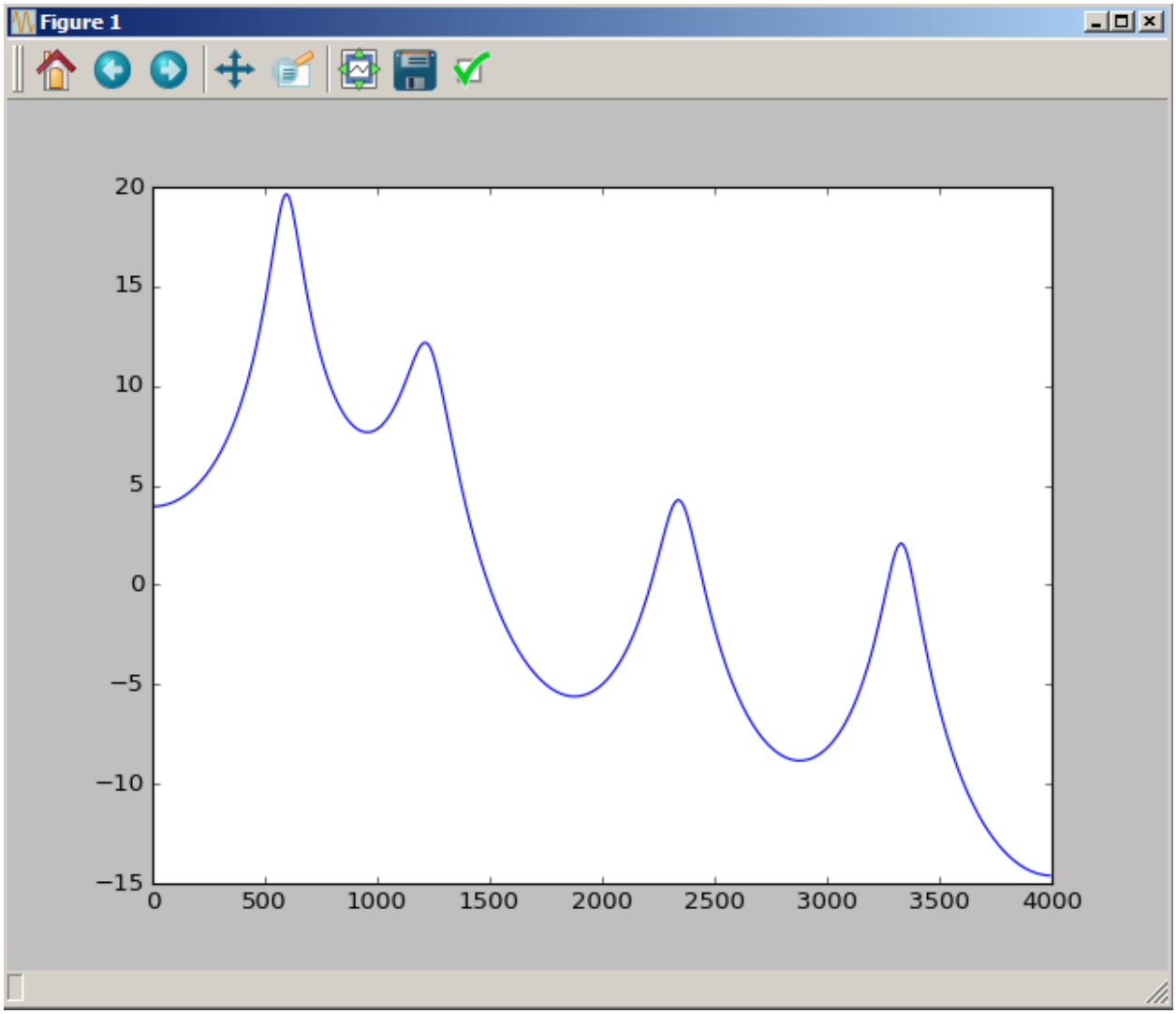

Рис. 4. Частотный отклик

Теперь вычислим частотный отклик с помощью функции `freqz` из пакета `scipy`:

```
w, h = scipy.signal.freqz(np.array([1]),np.hstack([1,-a]))
ax.plot(w/np.pi*Fs/2, 20*np.log10(np.abs(h)), 'r')
```

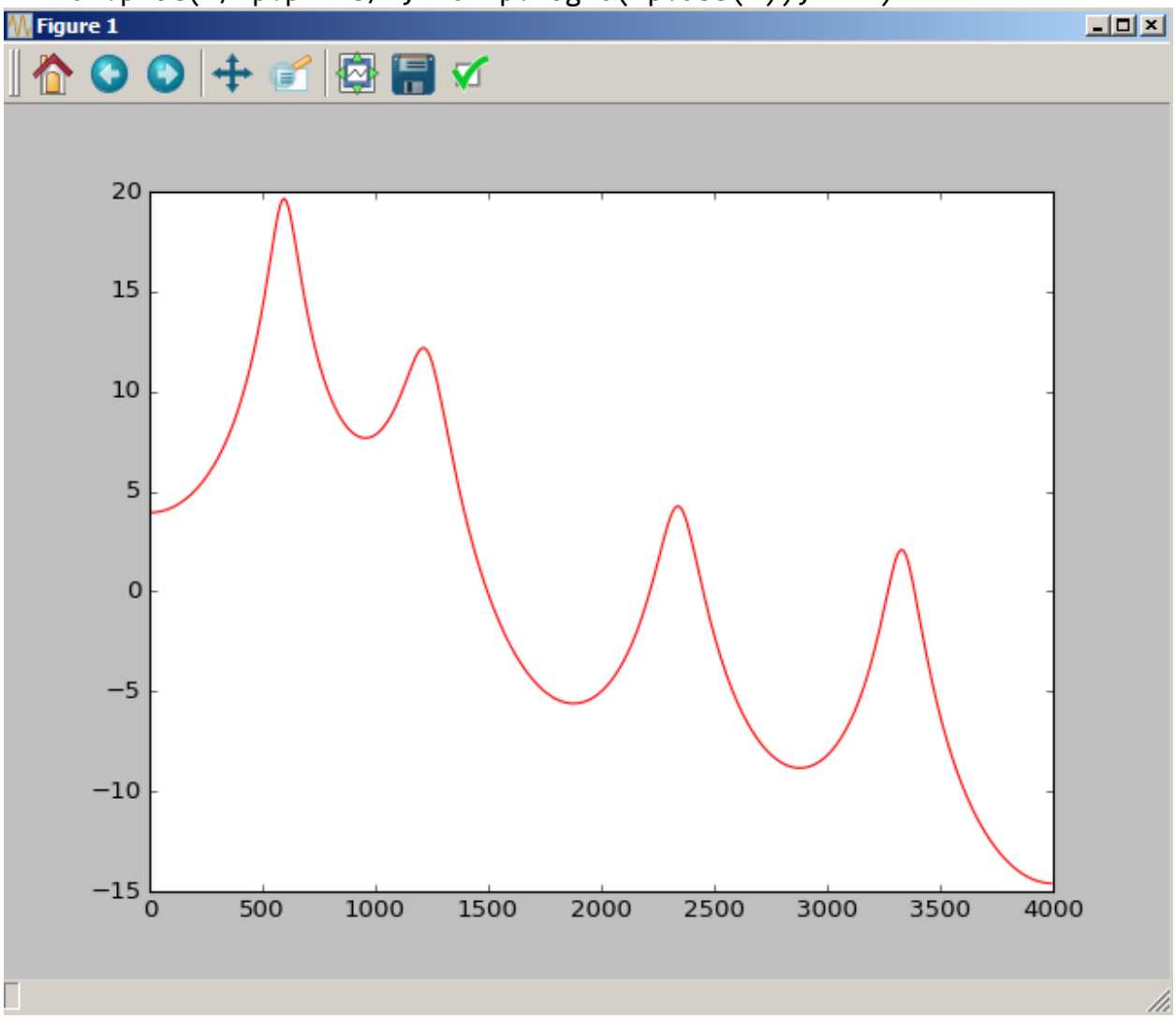

Рис. 5. Частотный отклик с помощью функции freqz

Сравним два варианта получения частотного отклика и увидим, что они одинаковы:

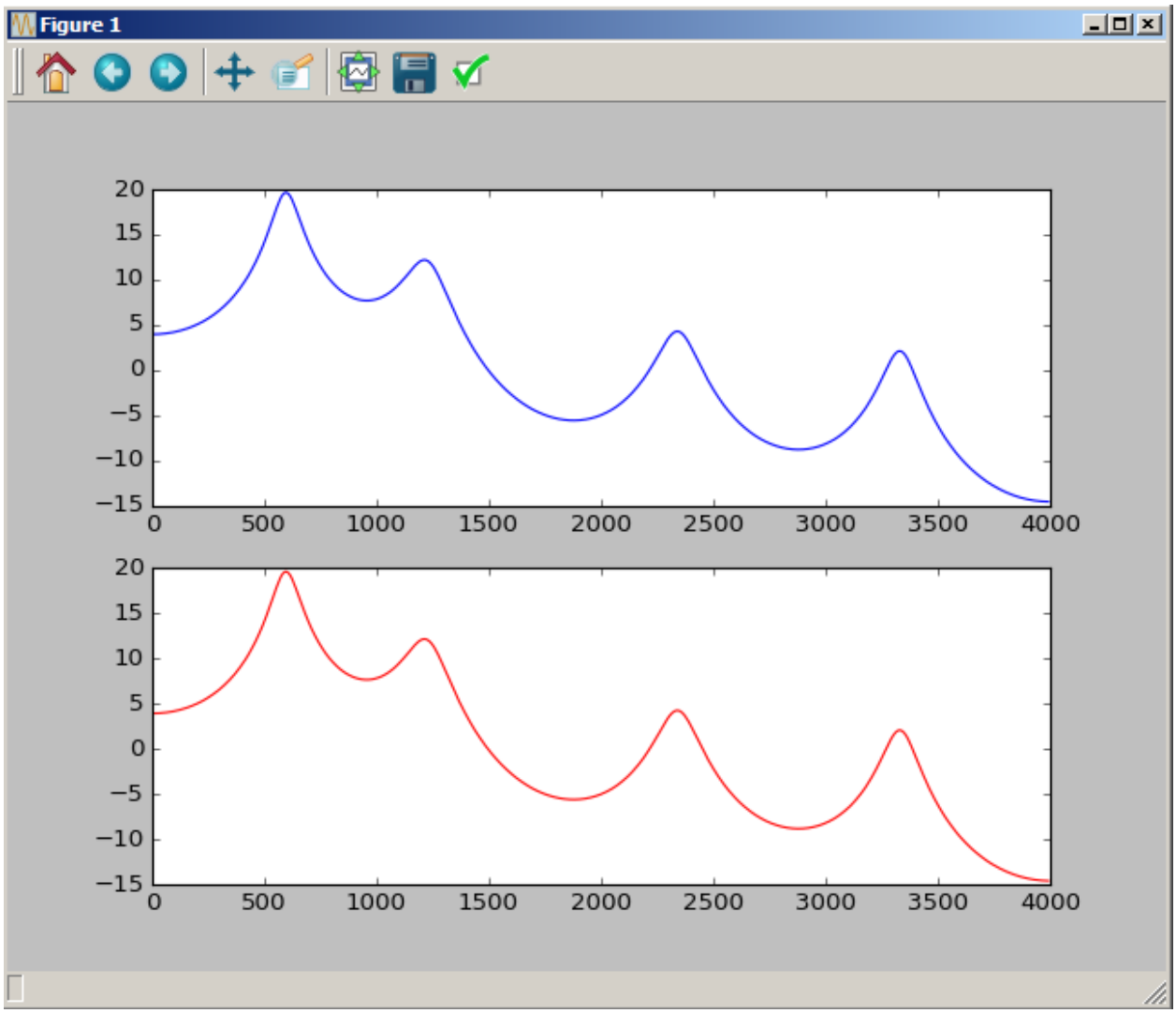

Рис. 6. Сравнение частотных откликов

Теперь исказим параметры и посмотрим, что происходит с сигналом:

1. Исказим параметры на 0.1:

```
w,h=scipy.signal.freqz(np.array([1]), np.hstack([1, -a]))
ax.plot(w/np.pi*Fs/2,20*np.log10(np.abs(h)),'b')

a1 = a + 0.1
w,h1=scipy.signal.freqz(np.array([1]),np.hstack([1,-a1]))
ax.plot(w/np.pi*Fs/2,20*np.log10(np.abs(h1)),'r')
```

Посчитаем спектральное искажение между исходным h и измененным h1:

```
    sd = np.sum(
(10*np.log10(np.abs(h)) - 10*np.log10(abs(np.abs(h1))))**2)/len(h)
```

При изменении параметров на 0.1, спектральное искажение между h и h1 получилось равным 2.04024893718.

Обернем расчеты спектрального искажения в функцию, чтобы проще было обращаться к ней:

```
def spectral_distortion(h, h1):
    return np.sum(
(10*np.log10(np.abs(h)) - 10*np.log10(abs(np.abs(h1))))**2 ) / len(h)
```

Теперь можем вызывать функцию подсчета спектрального искажения следующим образом:

```
    sd = spectral_distortion(h, h1)
```

2. Исказим параметры на 0.05:

```
    a_005 = a + 0.05
    w_005, h_005 = scipy.signal.freqz(
                        np.array([1]),np.hstack([1,-a_005]))
    ax.plot(w_005/np.pi*Fs/2, 20*np.log10(np.abs(h_005)),'r')
    sd_005 = spectral_distortion(h, h005)
```

sd_005 получился равным 2.02643318853.

3. Искажение параметров на 0.2:

```
    a_02 = a + 0.2
    w_02, h_02 = scipy.signal.freqz(
                       np.array([1]),np.hstack([1,-a_02]))
    ax.plot(w_02/np.pi*Fs/2, 20*np.log10(np.abs(h_02)),'r')
    sd_02 = spectral_distortion(h, h02)
```

sd_02 получился равным 6.3038777342.

Получили следующие отклонения:

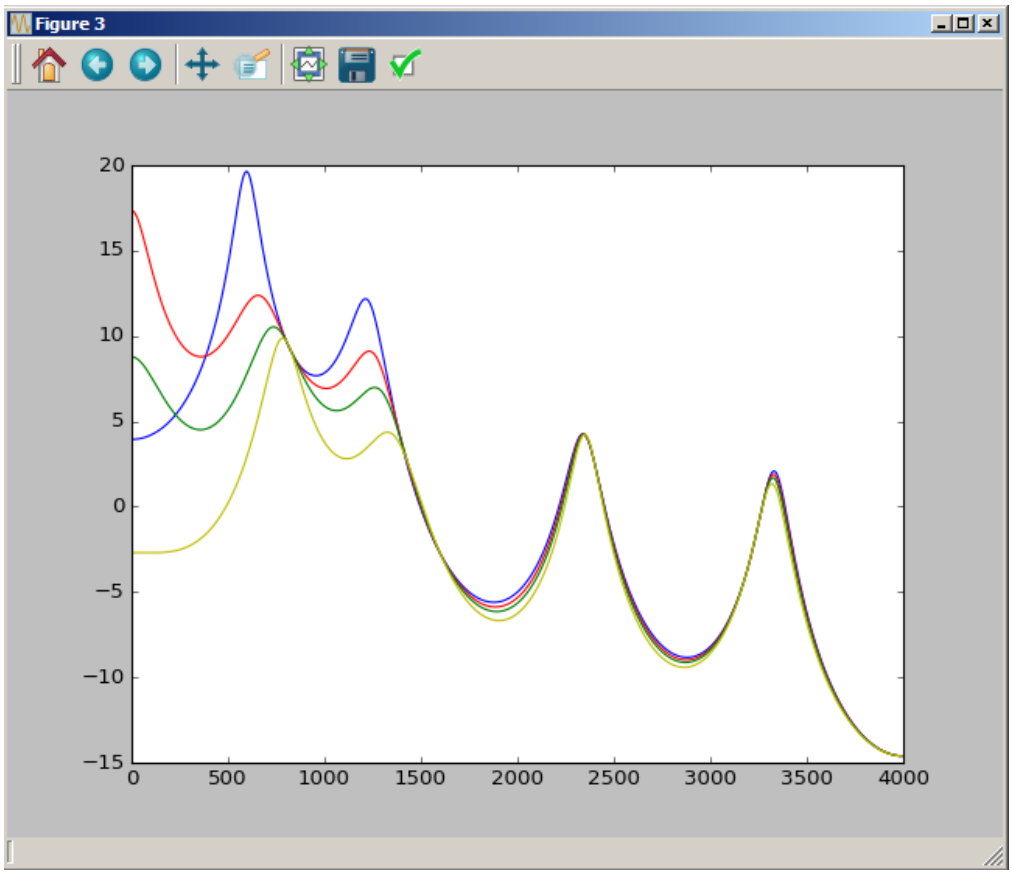

Рис. 7. Исходный и искаженные отклики

Вычислим теперь корни полинома и выделим положительные:

```
w,h=scipy.signal.freqz(np.array([1]),np.hstack([1,-a]))
ax.plot(w/np.pi*Fs/2,20*np.log10(np.abs(h)),'b')
```

Найдем корни с помощью функции roots из пакета numpy:

```
aa=np.roots(np.hstack([1, -a]))
```

Преобразуем получившийся результат так, чтобы получились единица измерения Герц:

```
ffreq = np.arctan2(aa.imag, aa.real)*Fs/(2*np.pi)
```

Выделим только положительные:

```
ffreq = ffreq[(ffreq>0)]
oy=20*np.log10(np.abs(h))
ax.stem(ffreq,np.max(oy)*np.ones(len(ffreq)))
ax.grid()
```

Посмотрим на результат:

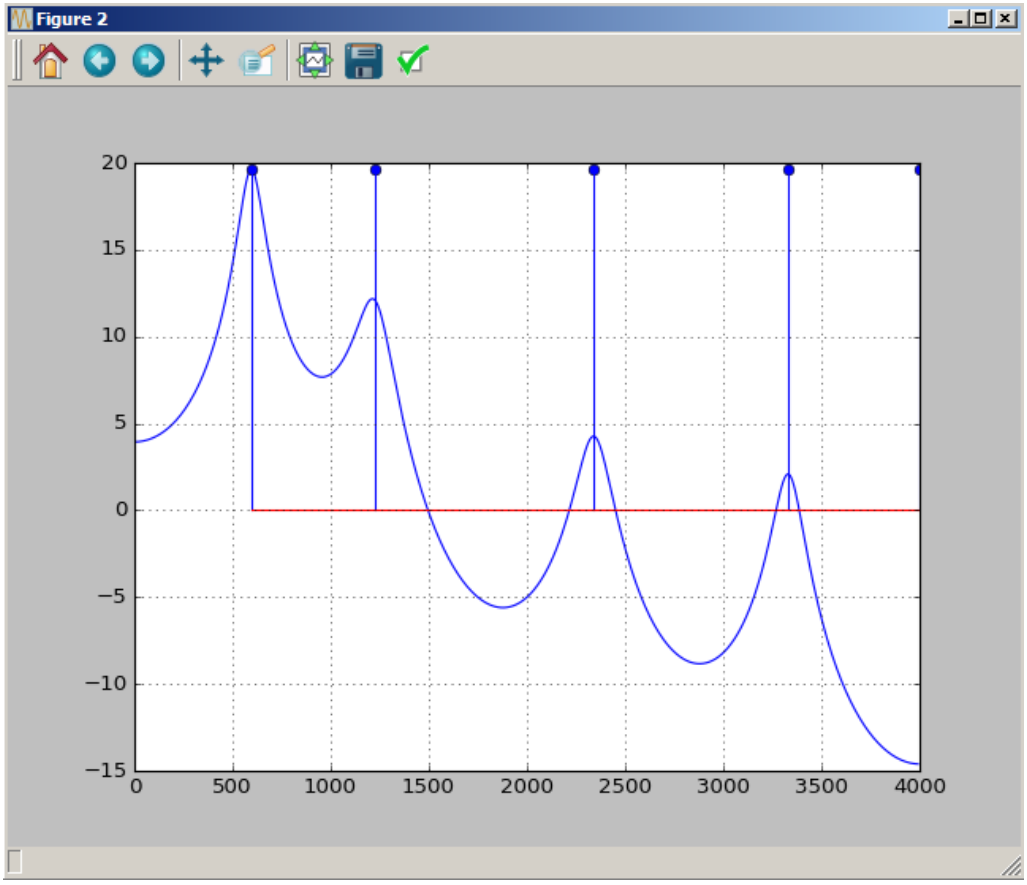

Рис. 8. Корни полинома

Рассмотрим алгоритм вычисления линейных спектральных параметров:

```
    pf,qf,lsf = lpc.generate_lsp(np.hstack([1,-a]),M)
    print('pf=',pf)
    print('qf=',qf)
    print('lsf1=',lsf)
>>>
pf= [ 1. ; -1.15871955 ;  0.24946477 ; 0.65698281 ; -0.10679032 ;
-0.00525006 ; -0.00525006 ; -0.10679032 ; 0.65698281 ; 0.24946477 ;
-1.15871955 ;  -1. ]
qf= [ 1. ; -1.09700179 ; 0.80916733 ; -0.95984375 ; 0.98493842 ;
-0.52711336 ; 0.52711336 ; -0.98493842 ; 0.95984375 ; -0.80916733 ;
1.09700179 ; -1. ]
lsf1= [ 0.06153832 ; 0.07657221 ; 0.10793556 ; 0.14935472 ; 0.1708007;
0.26167752 ; 0.29317542 ; 0.32781268 ; 0.40723917 ; 0.42328628]
```

Линейные спектральные частоты можем также получить:

```
    aa=np.hstack([1,-a])
    lsf = lpc.poly2lsf(aa)
    print('lsf2=',lsf)
>>>
lsf2= [ 0.06153832 ; 0.07657221 ; 0.10793556 ; 0.14935472 ; 0.1708007
;  0.26167752;
  0.29317542 ; 0.32781268 ; 0.40723917 ; 0.42328628],
```

сравнивая с `lsf1` видим, что элементы одинаковые.

Функция poly2lsf возвращает вектор lsf линейных спектральных частот, рассчитанный на основе вектора коэффициентов линейного предсказания.

Воспользуемся обратной функцией и сравним исходные данные с получившимися:

```
    aa_ = lpc.lsf2poly(lsf*(2*np.pi))
    print(aa)
    print(aa_)
>>>
[ 1.;  -1.12786067 ;  0.52931605 ; -0.15143047 ; 0.43907405 ;
-0.26618171;  0.26093165 ; -0.54586437 ; 0.80841328 ; -0.27985128 ;
-0.03085888]
[ 1. ; -1.12786067  ; 0.52931605 ; -0.15143047 ; 0.43907405 ;
-0.26618171; 0.26093165 ; -0.54586437 ; 0.80841328 ; -0.27985128 ;
-0.03085888]
```

Функция `lsf2poly` возвращает вектор, содержащий коэффициенты линейного предсказания, рассчитанные на основе вектора линейных спектральных частот `lsf`.

Рассмотрим квантизационную матрицу:

```
lspQ = np.array([
 [100, 170, 225, 250, 280, 340, 420, 500, 0, 0, 0, 0, 0, 0, 0, 0],
 [210, 235, 265, 295, 325, 360, 400, 440, 480, 520, 560, 610, 670,
  740, 810, 880],
 [420, 460, 500, 540, 585, 640, 705, 775, 850, 950, 1050, 1150,
  1250, 1350, 1450, 1550],
 [620, 660, 720, 795, 880, 970, 1080, 1170, 1270, 1370, 1470,
  1570, 1670, 1770, 1870, 1970],
 [1000, 1050, 1130, 1210, 1285, 1350, 1430, 1510, 1590, 1670,
  1750, 1850, 1950, 2050, 2150, 2250],
 [1470, 1570, 1690, 1830, 2000, 2200, 2400, 2600, 0, 0, 0, 0, 0,
  0, 0, 0 ],
 [1800, 1880, 1960, 2100, 2300, 2480, 2700, 2900, 0, 0, 0, 0, 0,
  0, 0, 0 ],
 [2225, 2400, 2525, 2650, 2800, 2950, 3150, 3350, 0, 0, 0, 0, 0,
  0, 0, 0 ],
 [2760, 2880, 3000, 3100, 3200, 3310, 3430, 3550, 0, 0, 0, 0, 0,
  0, 0, 0 ],
 [3190, 3270, 3350, 3420, 3490, 3590, 3710, 3830, 0, 0, 0, 0, 0,
  0, 0, 0 ]
])
```

Для каждого элемента из массива `lsf` делаем сравнение с элементами квантизационной матрицы и если элемент из массива `lsf` оказался наиболее близким к какому-то элементу из `lspQ`, то запоминаем номер строки.и формируем массив битов:

```
bits = np.array([3, 4, 4, 4, 4, 3, 3, 3, 3, 3])
findex = lpc.lsp_quant(lsf, M, bits, lspQ)
freq_q = lpc.findex2lsp(findex,lspQ)
```

Получаем коэффициенты линейного предсказания после квантования:

```
aq = lpc.lsf2poly(freq_q/Fs*(2*np.pi))
```

Построим частотную характеристику и оценим искажение:

```
    fig,ax = plt.subplots()
    w, h_ = scipy.signal.freqz(np.array([1]),np.hstack([1,-a]))
    ax.plot(w/np.pi*Fs/2,20*np.log10(np.abs(h_)),'b')

    w, h_1 = scipy.signal.freqz(np.array([1]), -aq)
    ax.plot(w/np.pi*Fs/2,20*np.log10(np.abs(h_1)),'r')

    sd_ = spectral_distortion(h_, h_1)
    print(sd_)
>>> sd_ = 0.648854763685
```

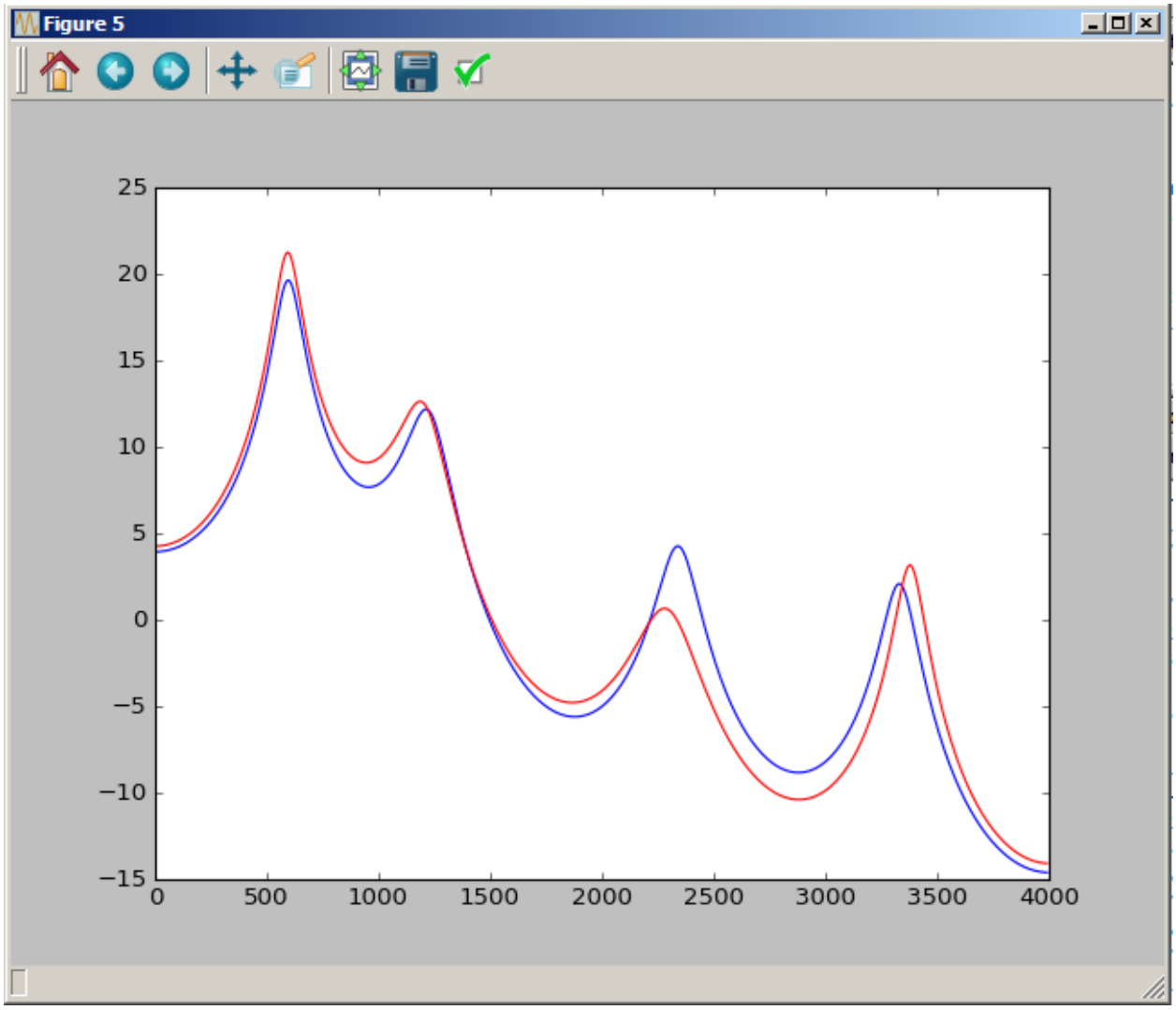

Рис. 9. Искажение

Синтезируем сигнал до и после квантования:

```
s_a = scipy.signal.lfilter(np.array([1]), np.hstack([1, -a]), e)
s_aq = scipy.signal.lfilter(np.array([1]), -aq, e)
```

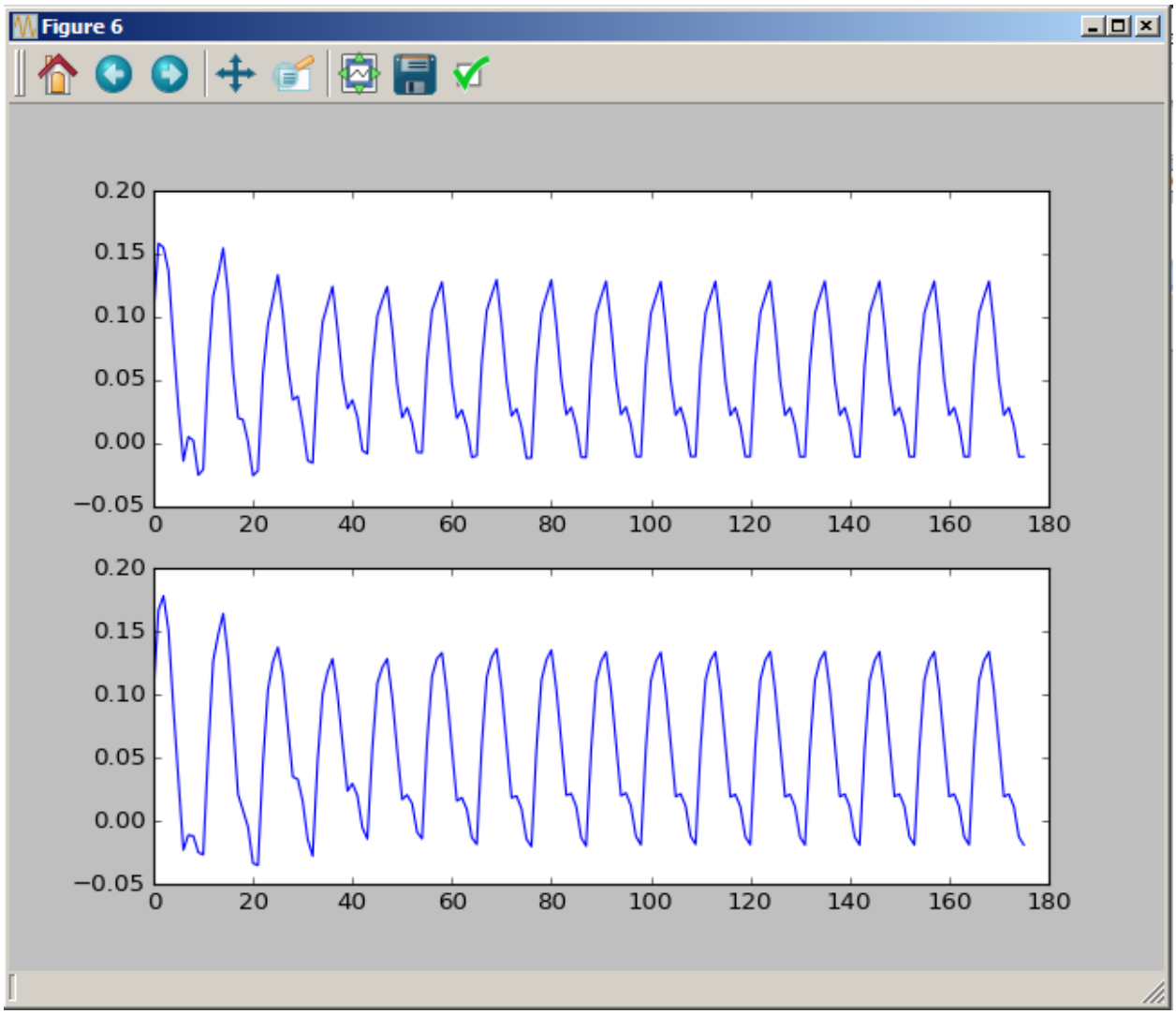

Рис. 10. Синтезированный сигнал до и после квантования

2. Другой вокализованный сегмент.

Рассмотрим сегмент, который начинается с 14105:

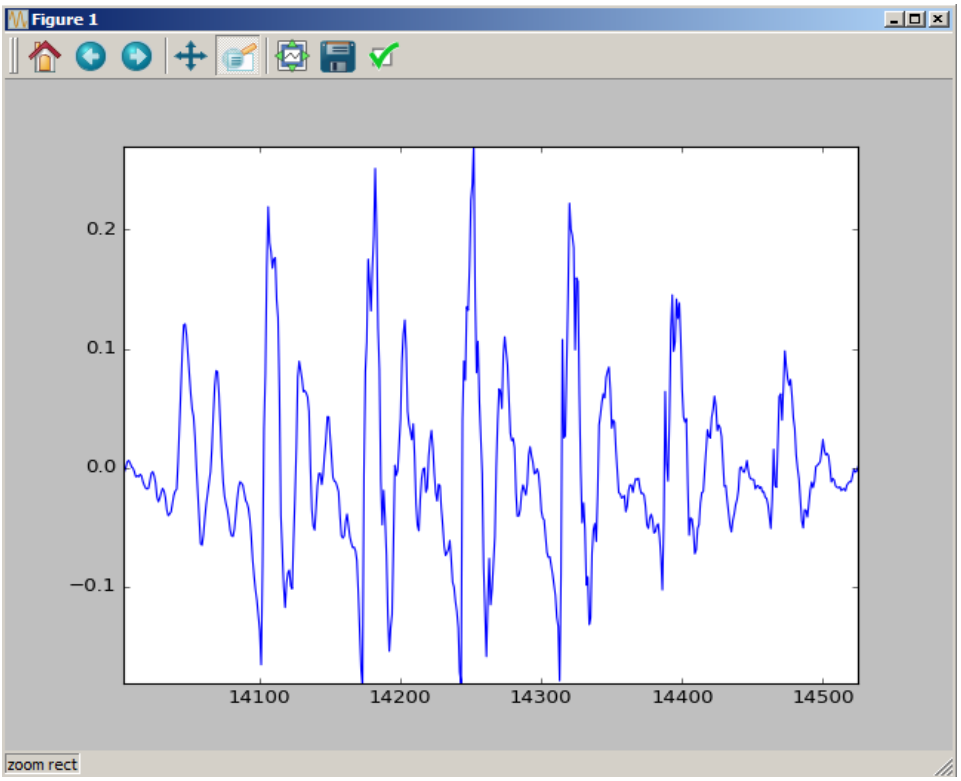

Рис. 11. Вокализованный сегмент

Выделим 160 отсчётов:

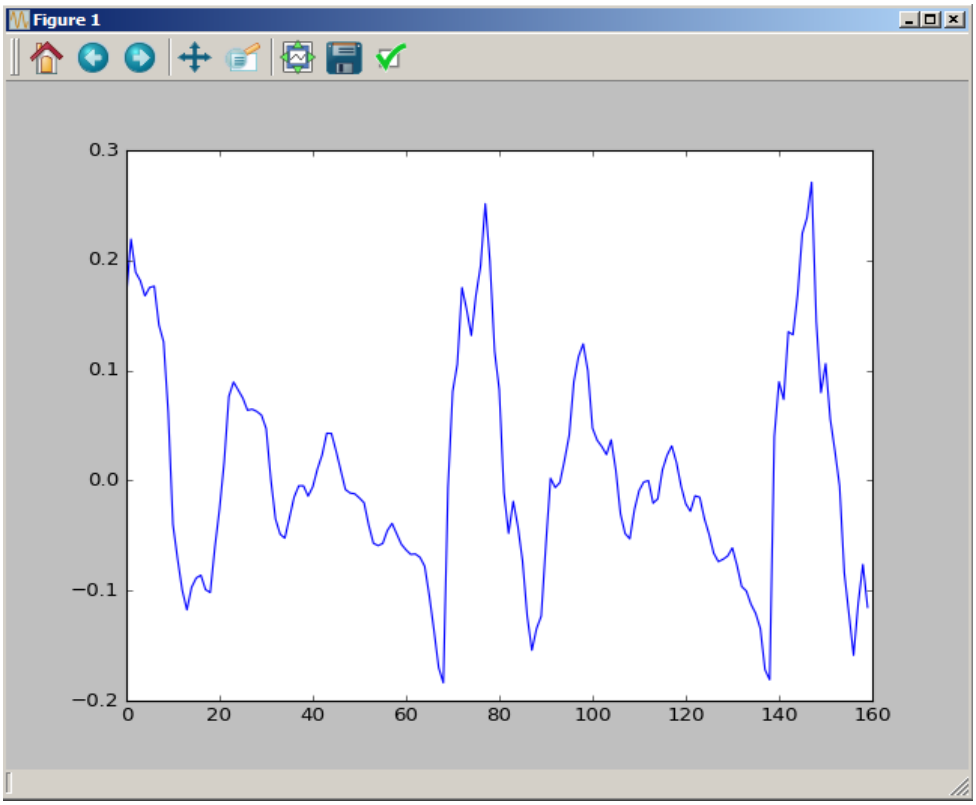

Рис. 12. Вокализованный сегмент(160 отсчётов)

Построим автокорреляционную функцию:

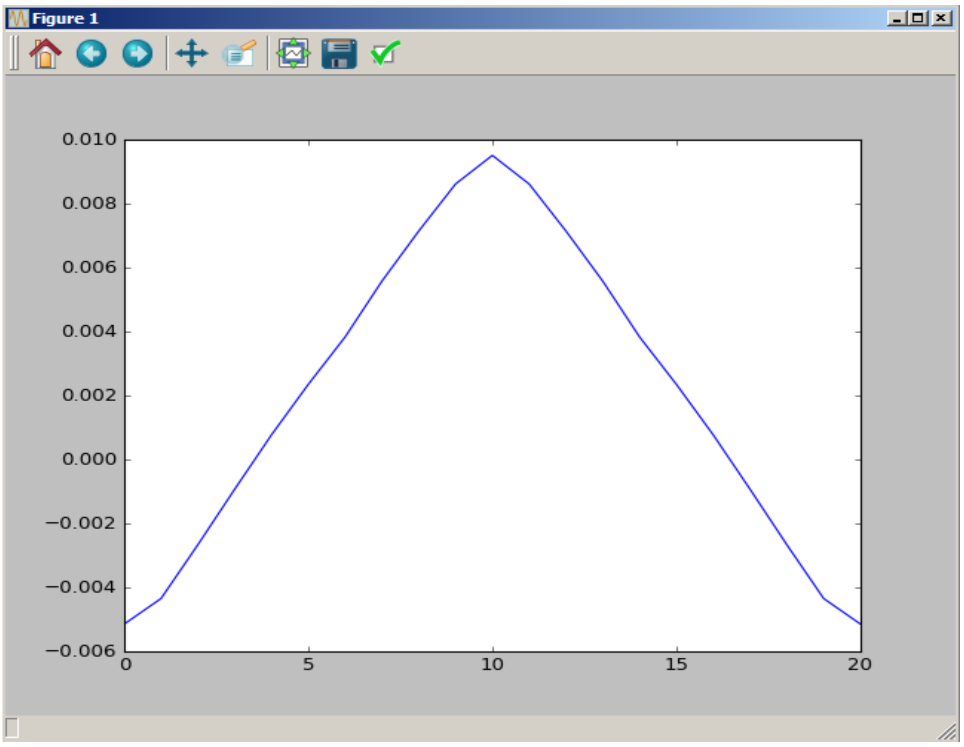

Рис. 13. Автокорреляционная функция

Вычисляем частотный отклик:

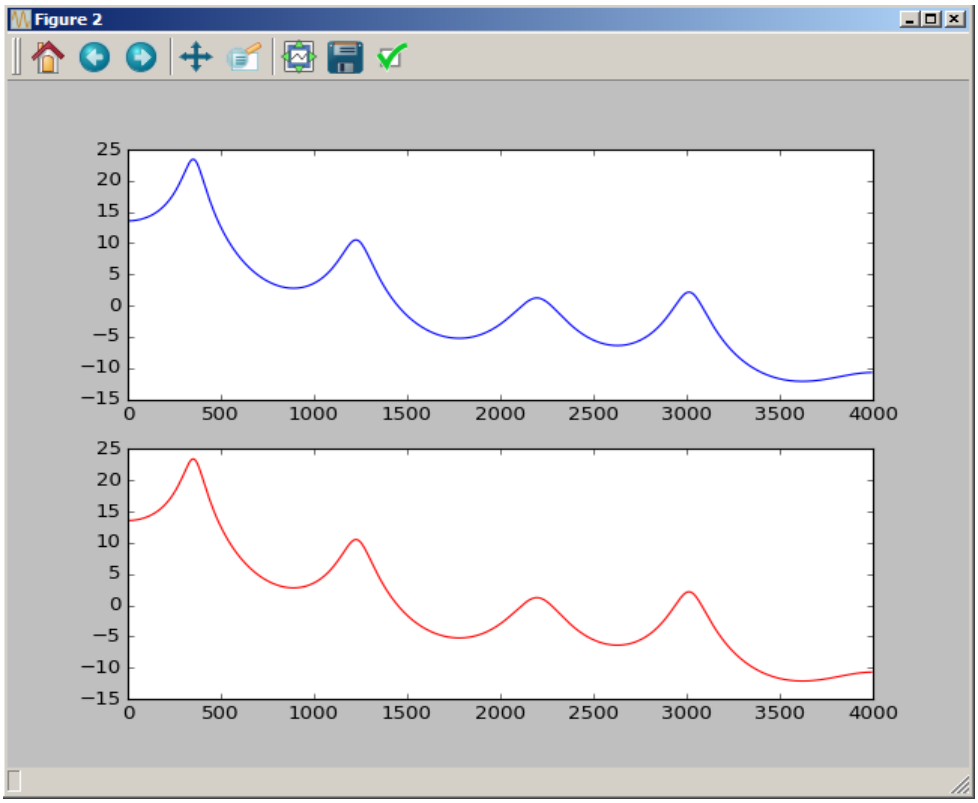

Рис. 14. Сравнение частотных откликов

Исказим параметры и посмотрим, как влияет изменение:

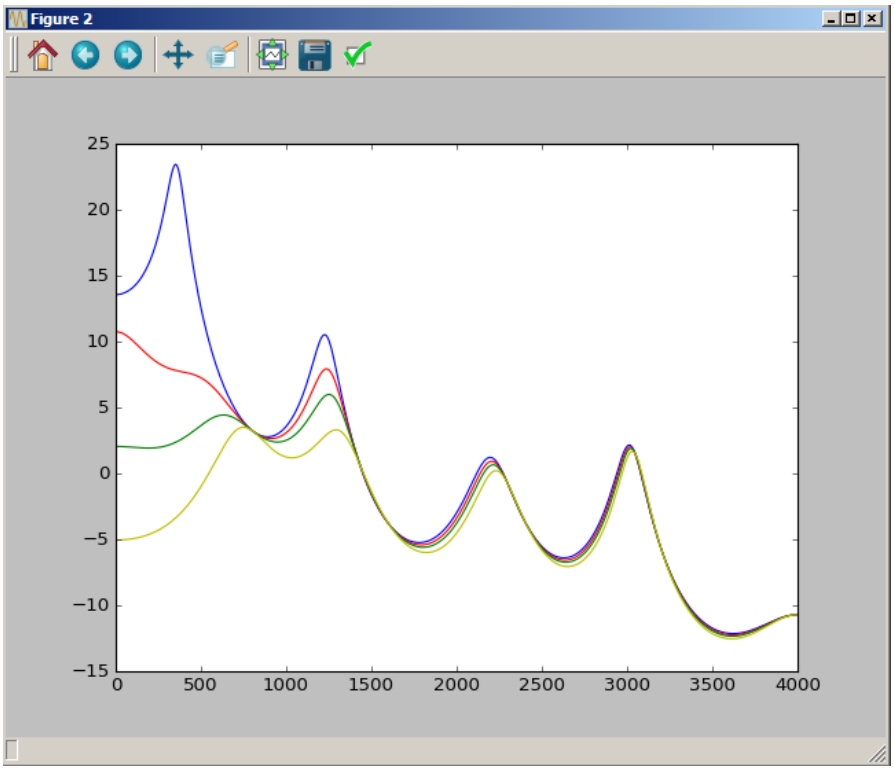

Рис. 15. Исходный и искаженные отклики

Считаем корни полинома:

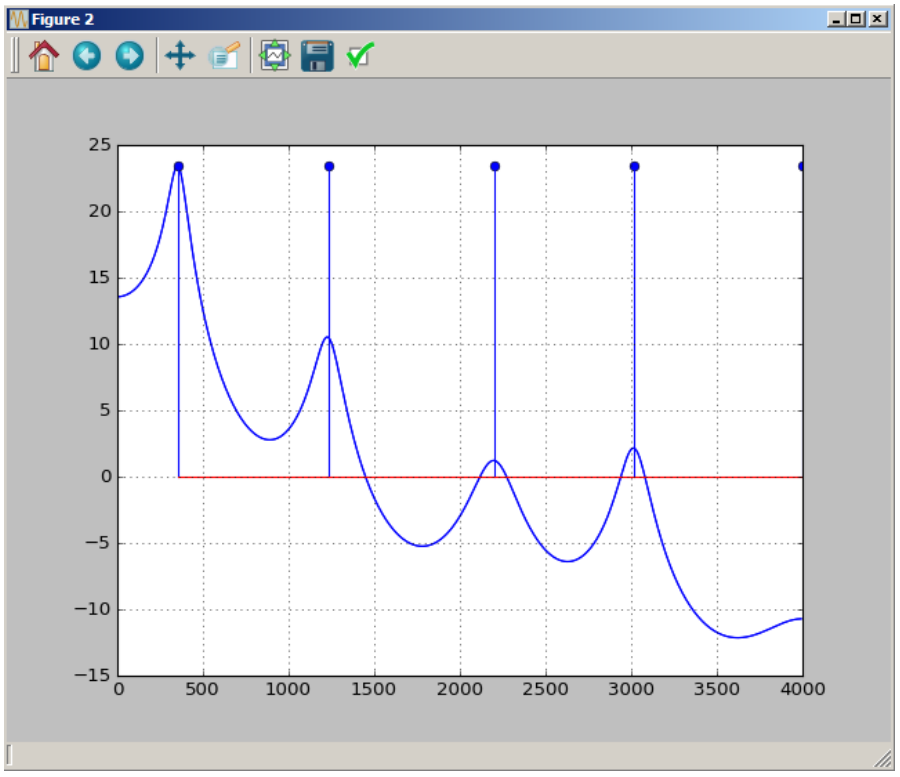

Рис. 16. Корни полинома

Вычислим линейные спектральные параметры:

```
pf= [ 1. -1.55521685  0.94754859 -0.33571506  0.39257714 -0.23923509
 -0.23923509  0.39257714 -0.33571506  0.94754859 -1.55521685  1.
]
qf= [ 1. -0.90074958  0.00632388 -0.42457246  0.59001106 -0.51146848
  0.51146848 -0.59001106  0.42457246 -0.00632388  0.90074958 -1.
]
lsf1= [ 0.03824816  0.04917077  0.10252782  0.15123963  0.17336698
0.25954968
  0.28639948  0.35671702  0.38015283  0.42909191]
lsf2= [ 0.03824816  0.04917077  0.10252782  0.15123963  0.17336698
0.25954968
  0.28639948  0.35671702  0.38015283  0.42909191]
```

Вычислим коэффициенты линейного предсказания до и после квантования:

```
aa =  [ 1. -1.22798322  0.47693623 -0.38014376  0.4912941  -0.37535178
  0.1361167  -0.09871696  0.0444287   0.47061236 -0.32723364]
aa_ =  [ 1. -1.22798322  0.47693623 -0.38014376  0.4912941  -
0.37535178
  0.1361167  -0.09871696  0.0444287   0.47061236 -0.32723364]
aq =  [ 1. -1.38122883  0.69516973 -0.47591087  0.61278629 -0.51290072
  0.14635192 -0.03915901 -0.01958667  0.39000495 -0.23737675]
```

Оценим искажение:

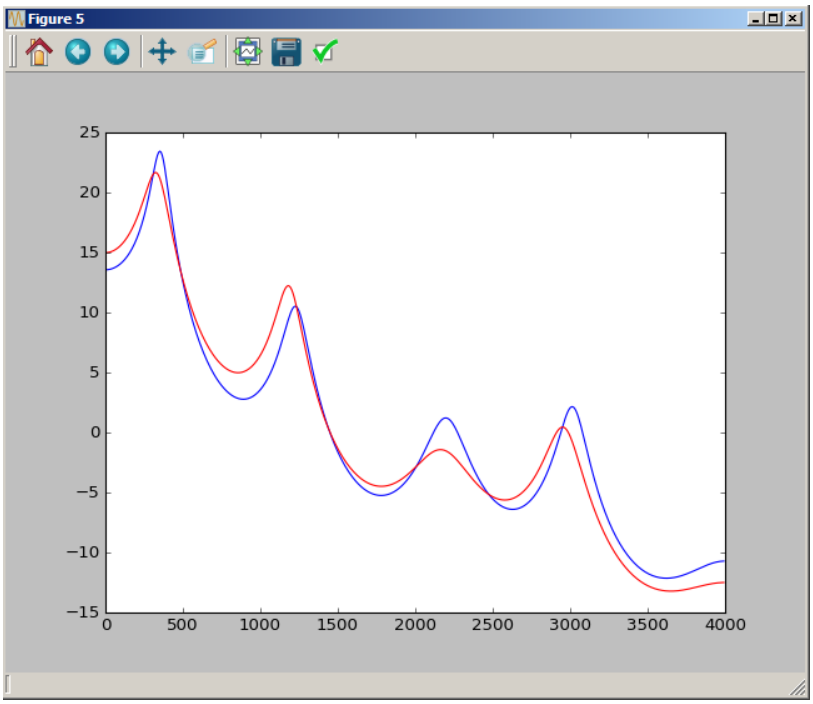

Рис. 17. Искажение

`sd_ = 0.670742057706.`

Синтезируем сегмент сигнала по коэффициентам линейного предсказания до и после квантования:

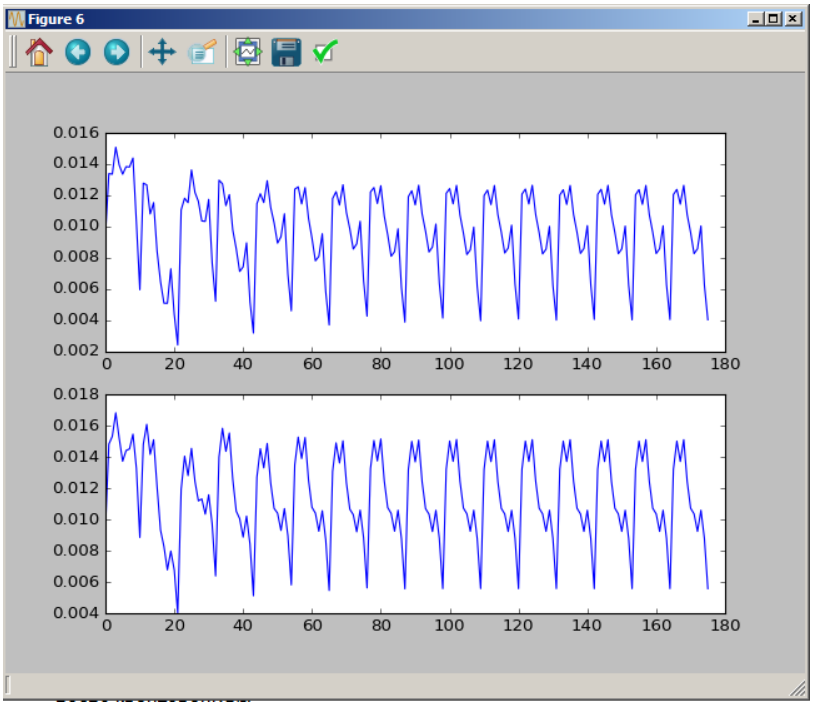

Рис. 18. Синтезированный сигнал до и после квантования

3. Невокализованный сегмент

Выберем невокализованный сегмент, например с 12130 отсчёта:

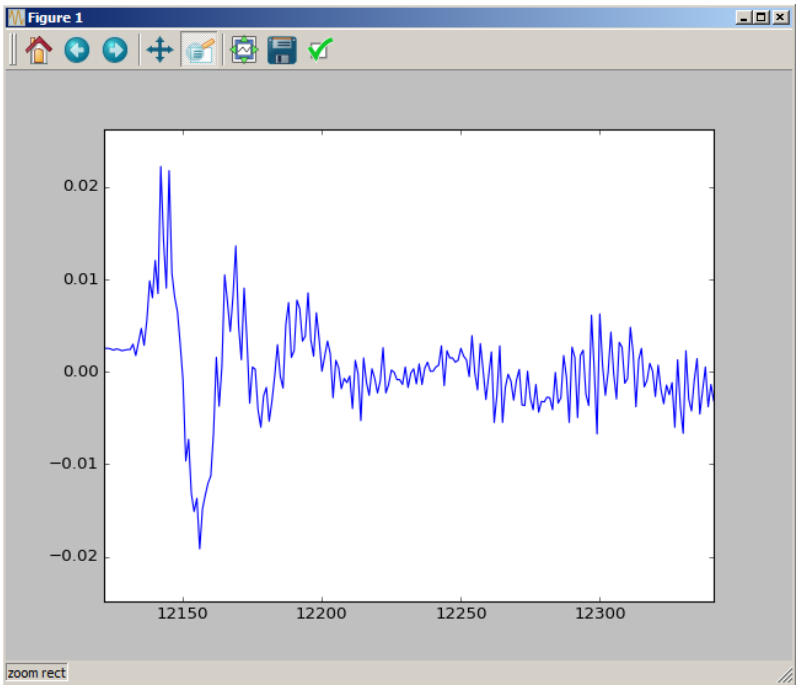

Рис. 19. Невокализованный сегмент

Вычислим автокорреляционную функцию:

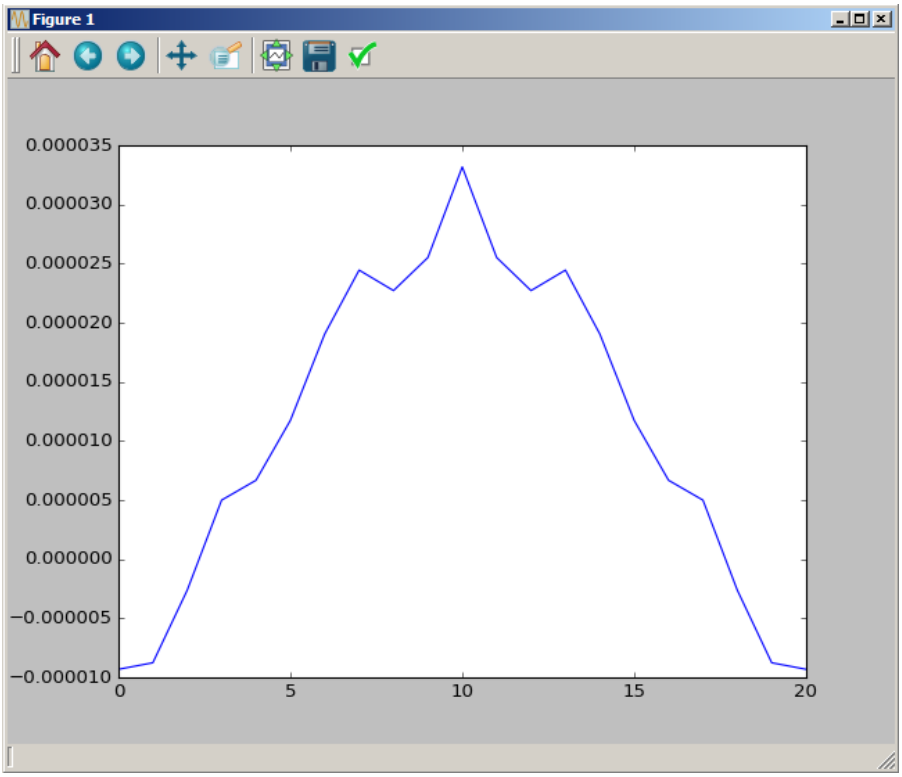

Рис. 20. Автокорреляционная функция для невокализованного сегмента

Построим частотные отклики:

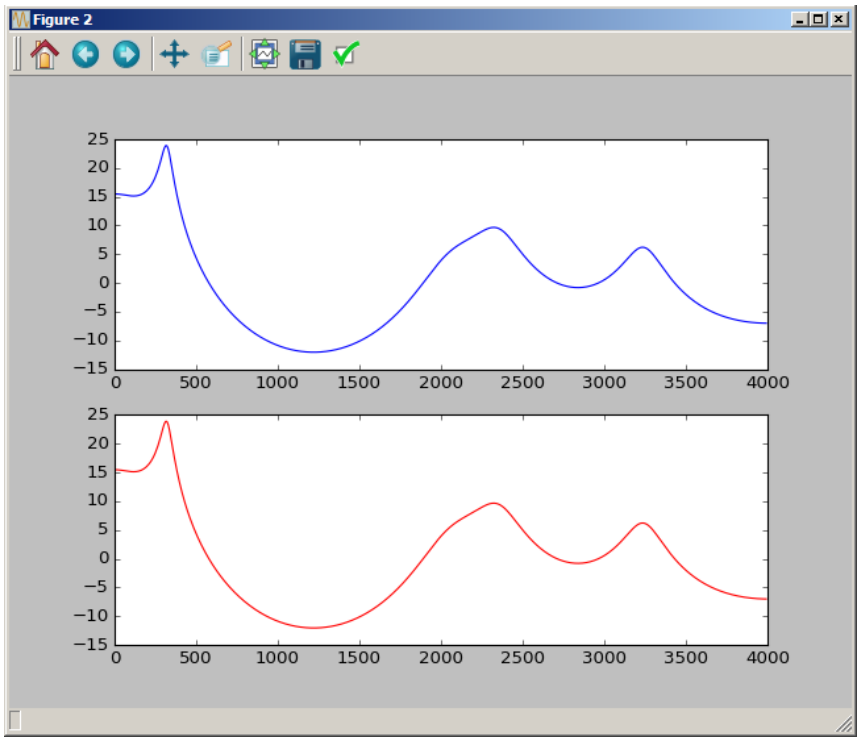

Рис. 21. Сравнение частотных откликов невокализованного сегмента

Искажаем параметры:

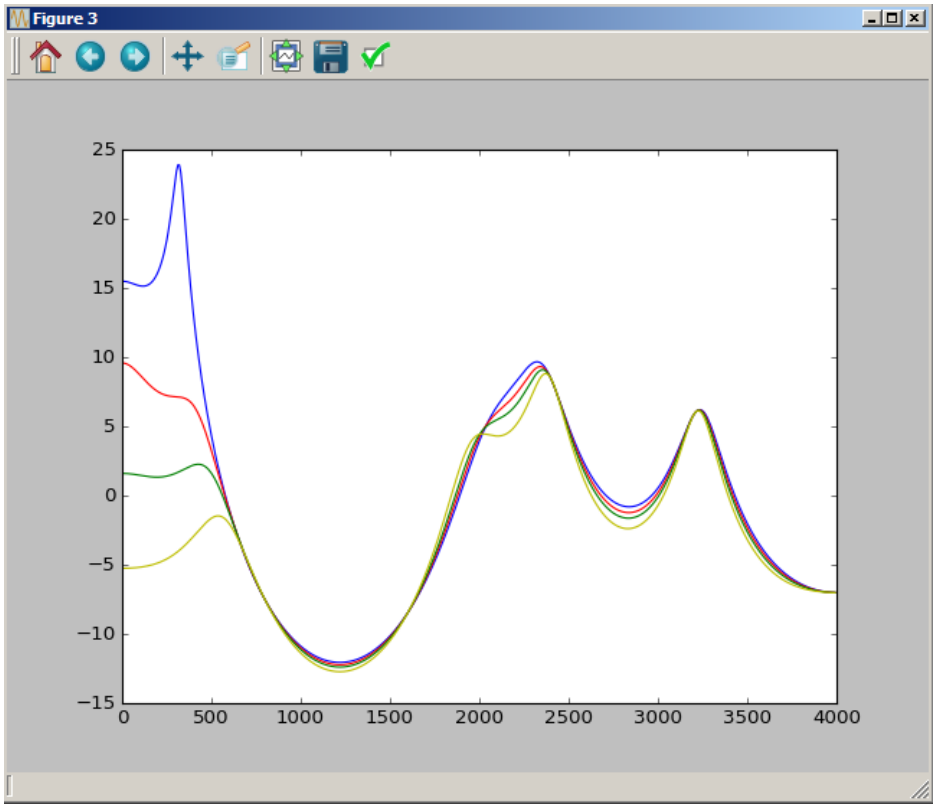

Рис. 22. Искажение параметров невокализованного сегмента

Находим корни полинома:

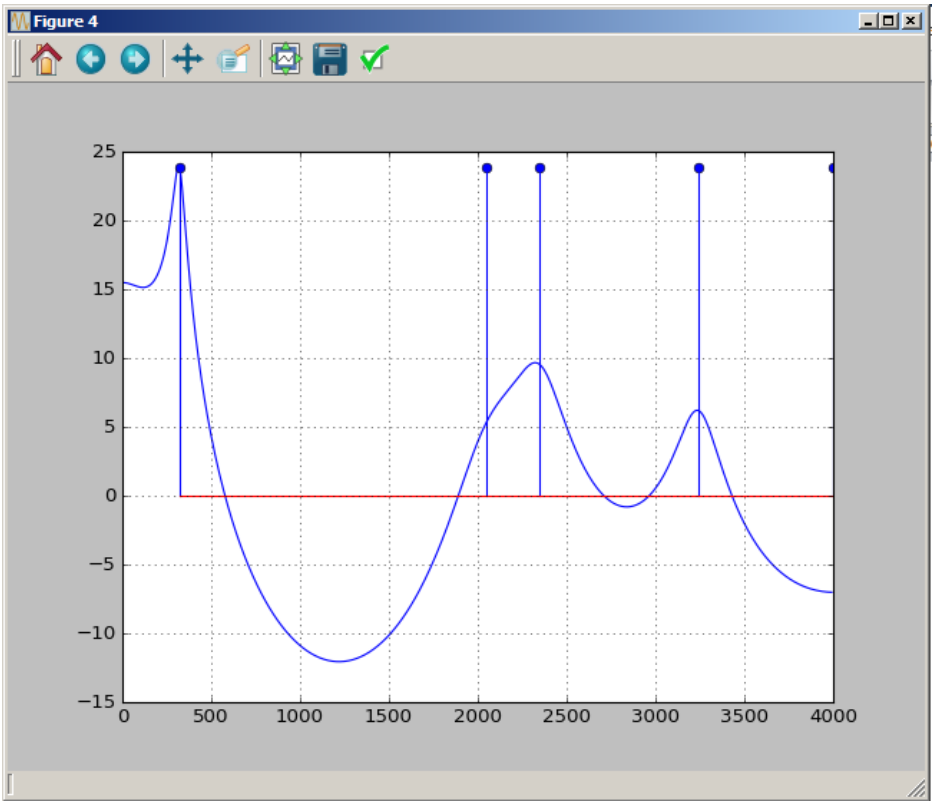

Рис. 23. Корни полинома

Вычисляем линейные спектральные параметры:

```
pf= [ 1. -0.36750933 -0.52210656 -0.96875864 -0.10140517  1.12788678
1.12788678 -0.10140517 -0.96875864 -0.52210656 -0.36750933  1. ]
qf= [ 1. 0.09754076  0.00722851 -1.21190723 -0.66790625
-0.78884854 0.78884854  0.66790625  1.21190723 -0.00722851 -0.09754076
-1. ]
lsf1= [ 0.03195916  0.04114896  0.07307802  0.21300987  0.25636283
0.28299112 0.30459233  0.36092291  0.40198381  0.42755254]
lsf2= [ 0.03195916  0.04114896  0.07307802  0.21300987  0.25636283
0.28299112 0.30459233  0.36092291  0.40198381  0.42755254]
```

Вычисляем коэффициенты линейного предсказания до и после квантования:

```
aa_ =  [ 1.  -0.13498428 -0.25743903 -1.09033293 -0.38465571
0.16951912 0.95836766  0.28325054  0.1215743  -0.26466753 -0.23252504]
aq =  [ 1.  -0.15130262 -0.32456947 -1.04162014 -0.30290684
0.15482808  0.97504628  0.26483769 -0.00275711 -0.23748929
-0.17005126]
sd_ = 0.448581841971
```

Оцениваем искажение:

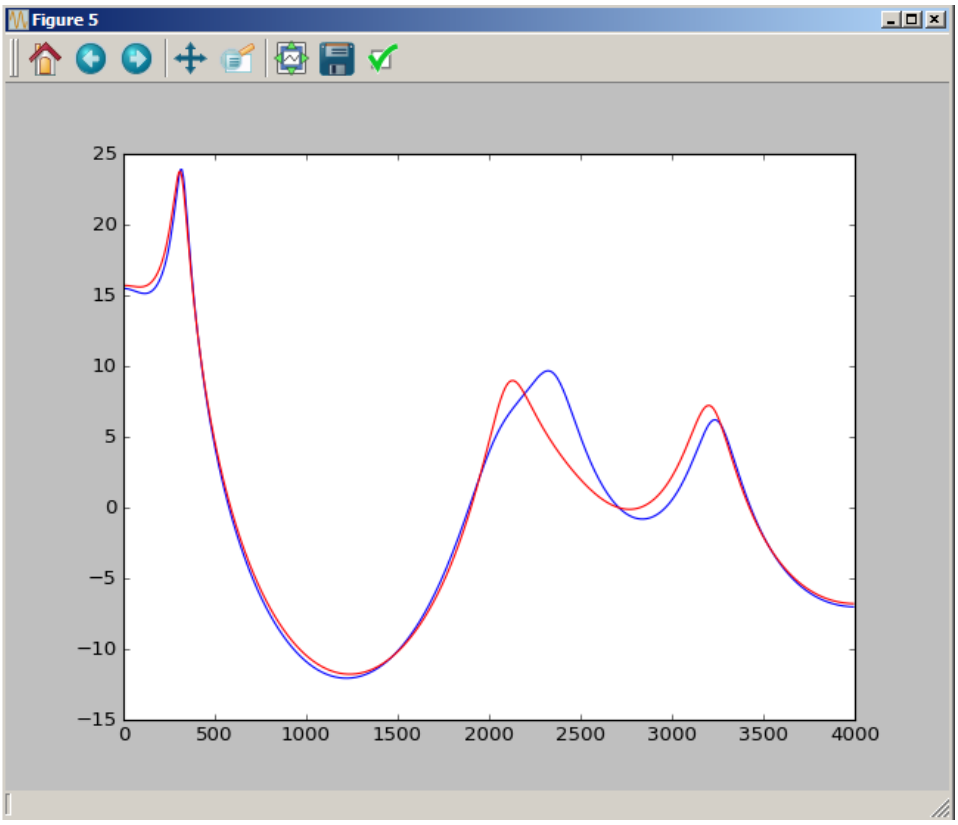

Рис. 24. Искажение

Для невокализованного сегмента сгенерируем «белый шум» и синтезируем сегмент сигнала по коэффициентам линейного предсказания до и после квантования:

```
    e_wn = np.sqrt(30)*np.random.randn(160)
    s_a = scipy.signal.lfilter(np.array([1]), np.hstack([1, -a]),
e_wn)
    s_aq = scipy.signal.lfilter(np.array([1]), -aq, e_wn)
```

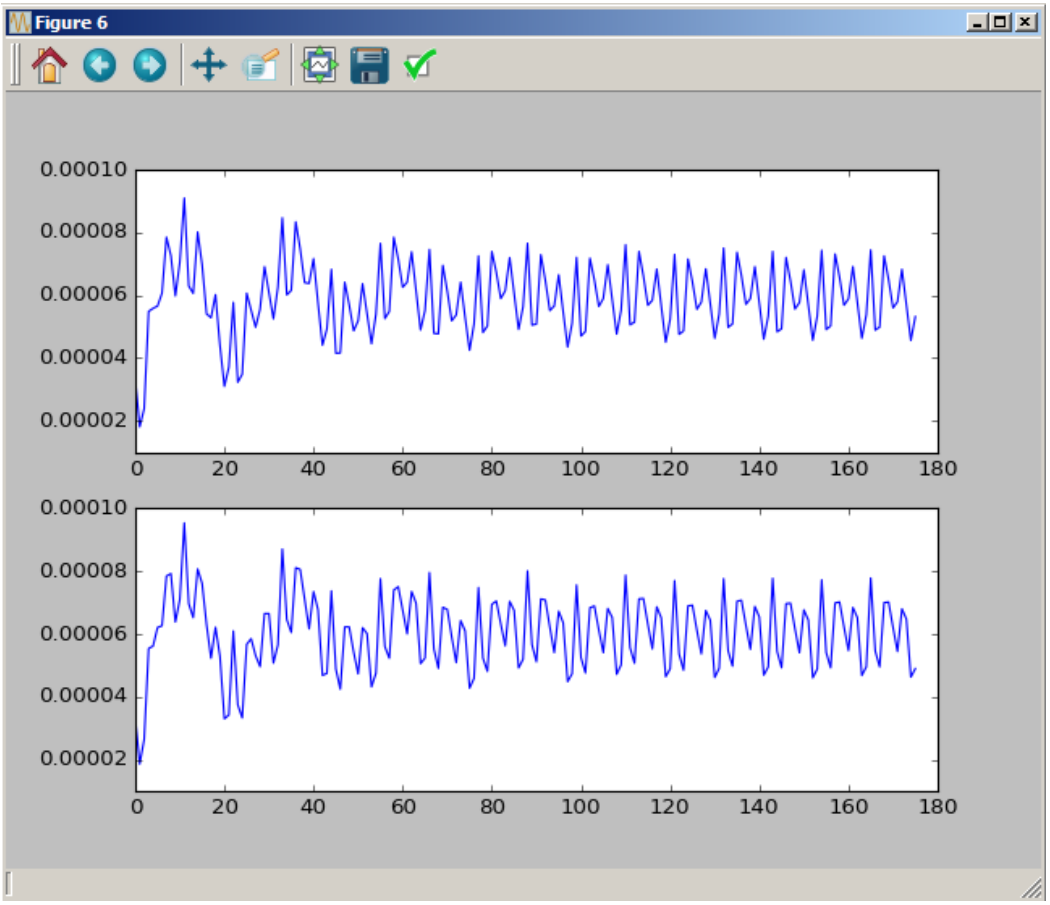

Рис. 25. Синтезированный сигнал до и после квантования

4. Вычисления для M = 3.

Теперь вычислим для вокализованного сигнала с M = 3:

Автокорреляционную функцию:

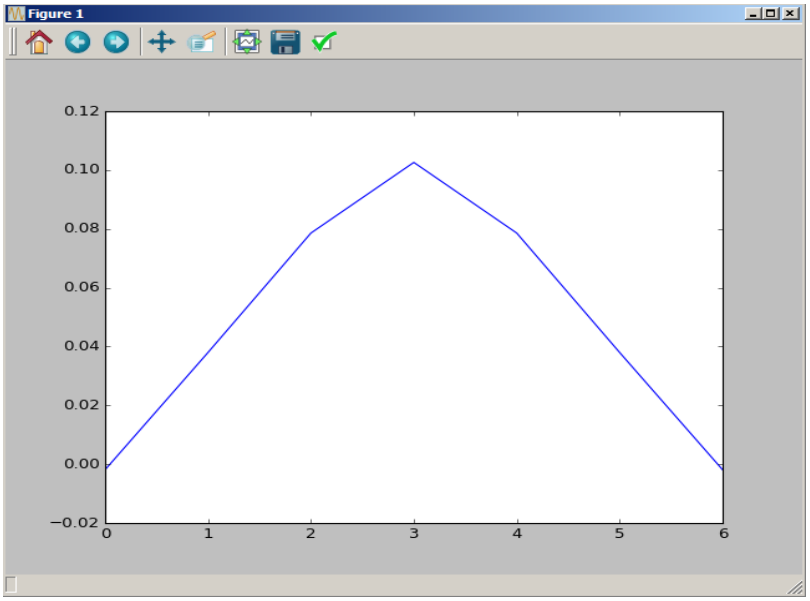

Рис. 26. Автокорреляционная функция

Частотные отклики:

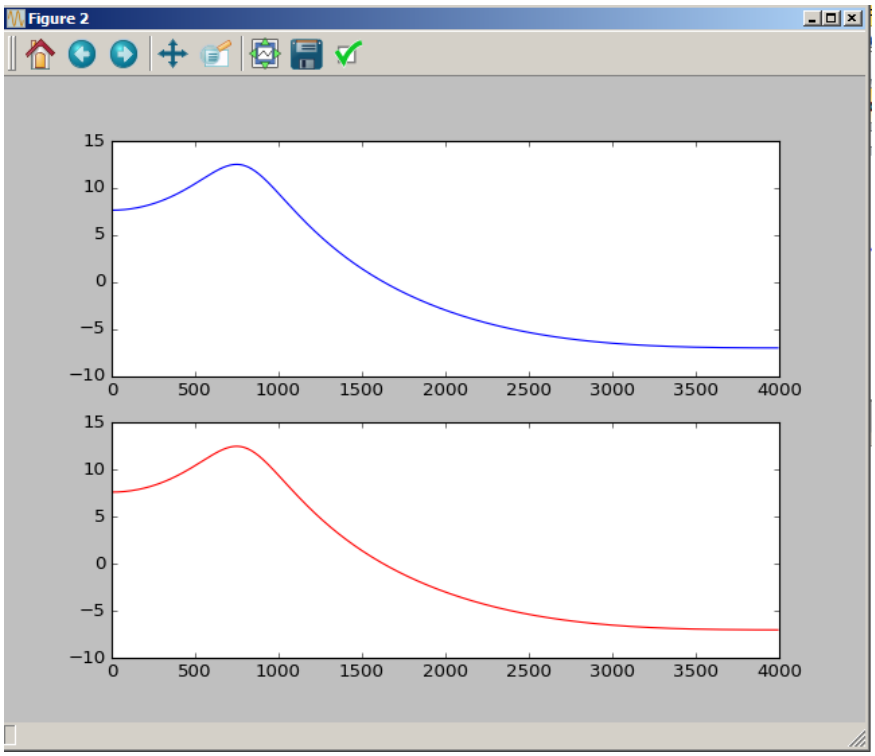

Рис. 27. Сравнение частотных откликов

Искажения параметров:

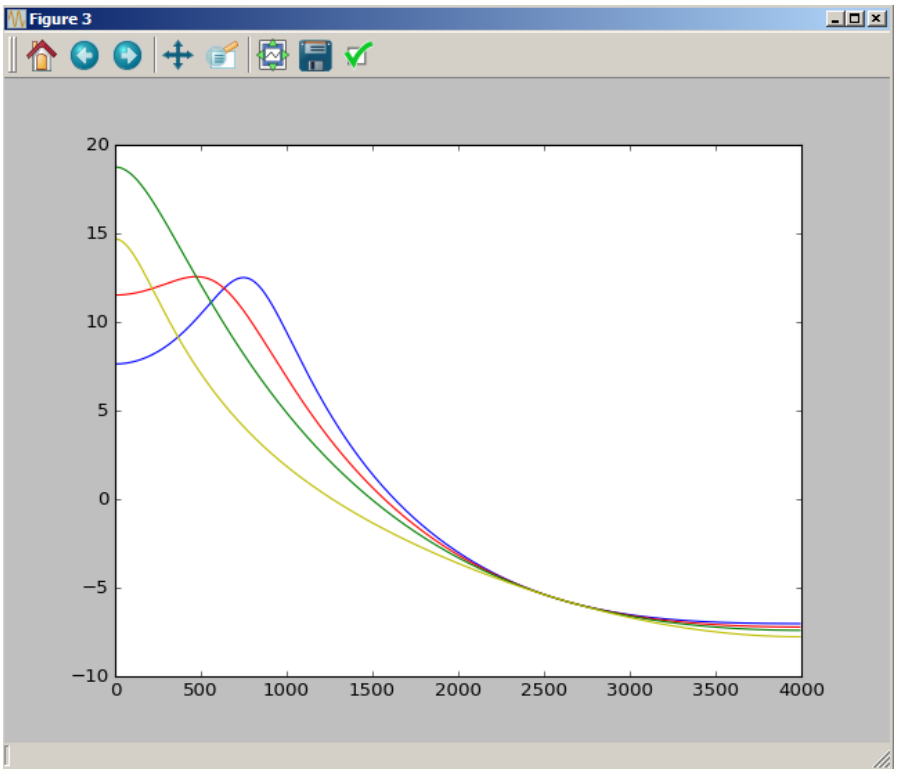

Рис. 28. Искажение параметров

Корни полинома:

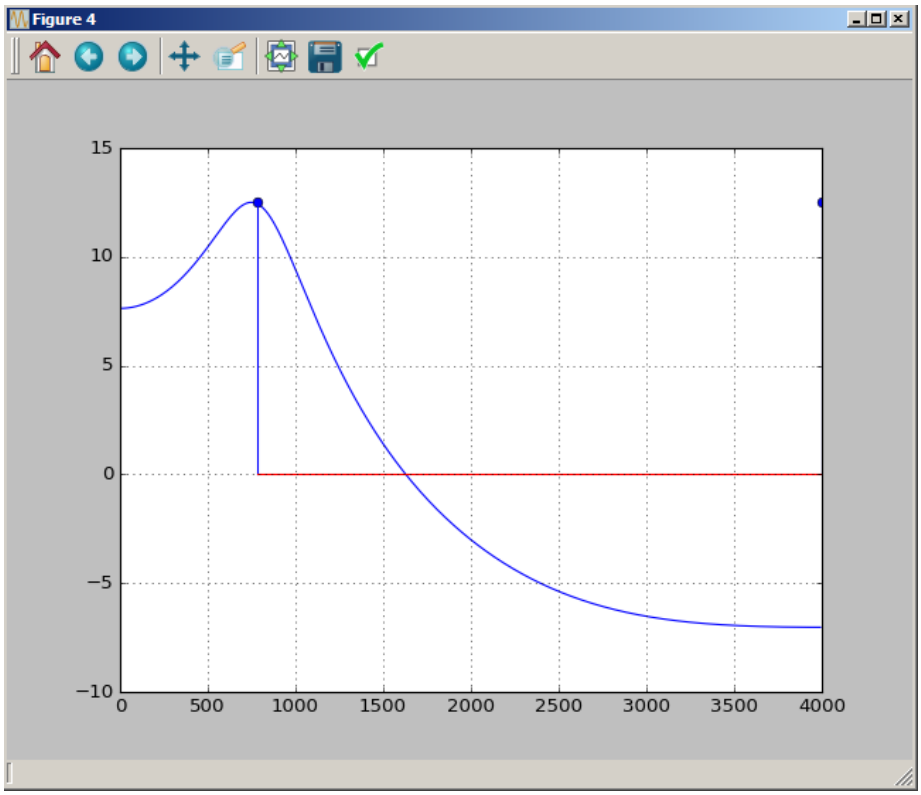

Рис. 29. Корни полинома