Hilbert变换提取信号特征的Python实现

开篇点题：

希尔伯特变换(hilbert transform) 一个连续时间信号s(t)的希尔伯特变换等于该信号通过具有冲激响应h(t)=1/πt的线性系统以后的输出响应sh(t)。

好的，这是Hilbert变换的定义，我们这里讨论它的一个具体用途，提取信号特征值，提取信号特征值有什么用呢？

先来一段特征值的定义：

设 A 是n阶方阵，如果存在数m和非零n维列向量 x，使得 Ax=mx 成立，则称 m 是A的一个特征值或本征值。

所以信号特征值可以用来代替一段信号对系统产生的影响，或者说取代一段信号并填补它功能的空缺，我们按照这个思想，在外界或内部改变信号的某些条件之后，可以用特征值随外界或内部因素的变化图像来反映干扰因素对信号的影响，或者反映信号本身对外界干扰的抵抗力及自身的稳定性、鲁棒性等性质。

那么具体怎么用呢？设有一个实值函数s(t)，其希尔伯特反变换记作(t)，则有：

希尔伯特变换：

E:\Signal Feature Extraction\希尔伯特变换.PNG

希尔伯特反变换：

E:\Signal Feature Extraction\希尔伯特反变换.PNG

上面说的就是初始信号s(t)，我们首先把它做一次希尔伯特变换，然后提取特征值。

都有什么特征值呢？

那我们得先补充一个知识点，包络：随机过程的振幅随着时间变化的曲线。也就是信号的振幅与时间（A-t）间的函数关系。

那么根据不同信号包络上高阶统计量特征不同，可分为R值特征和J值特征。

信号包络R值特征：R值是信号包络的方差与包络均值的平方的比值：

信号包络J值特征：J值是信号包洛的四阶矩和二阶矩平方之间的差值与包络均值的平方的比值：

双谱特征：利用积分的方法将二维双谱积分函数转换为一维双谱积分函数，从而实现计

算方法简单的积分双谱：

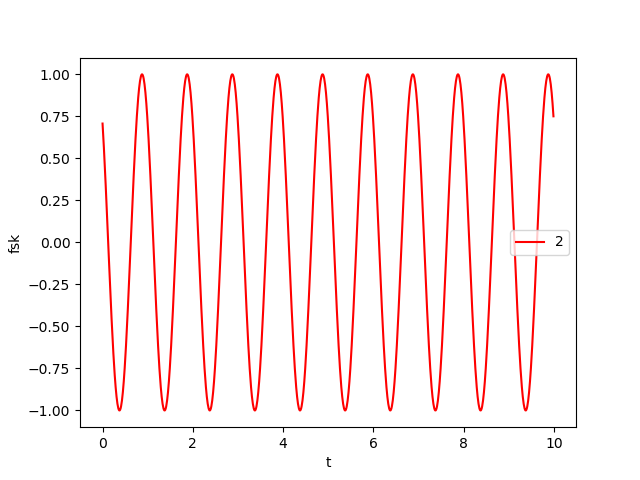
根据辐射源信号指纹识别技术<http://xueshu.baidu.com/usercenter/paper/show?paperid=f2c7b987bd3af320909da0e1c09a723b&site=xueshu_se>，这篇论文，三种特征值能够很好的表现信号性能。

好了，基础的知识就是这些，下面我们写代码实现Hilbert变换，计算并提取三种特征值。

1. **import** numpy as np
2. **from** math **import** pi
3. **import** matplotlib.pyplot as plt
4. **import** math
5. **from** scipy **import** fftpack
6. **from** sklearn **import** preprocessing
7. **import** neurolab as nl
9. # 码元数
10. size = 10
11. sampling\_t = 0.01
12. t = np.arange(0, size, sampling\_t)
14. #随机生成信号序列
15. a = np.random.randint(0, 2, size)   #产生随机整数序列
17. m = np.zeros(len(t), dtype=np.float32)    #产生一个给定形状和类型的用0填充的数组
18. **for** i **in** range(len(t)):
19. m[i] = a[int(math.floor(t[i]))]
21. ts1 = np.arange(0, (np.int64(1/sampling\_t) \* size))/(10\*(m+1))
22. fsk = np.cos(np.dot(2 \* pi, ts1) + pi / 4)
24. **def** awgn(y, snr):      #snr为信噪比dB值
25. snr = 10 \*\* (snr / 10.0)
26. xpower = np.sum(y \*\* 2) / len(y)
27. npower = xpower / snr
28. **return** np.random.randn(len(y)) \* np.sqrt(npower) + y
30. **def** feature\_rj(y):         #[feature1, f2, f3] = rj(noise\_bpsk, fs)
31. h = fftpack.hilbert(y)   # hilbert变换
32. z = np.sqrt(y\*\*2 + h\*\*2)   # 包络
33. m2 = np.mean(z\*\*2)     # 包络的二阶矩
34. m4 = np.mean(z\*\*4)     # 包络的四阶矩
35. r = abs((m4-m2\*\*2)/m2\*\*2)
36. Ps = np.mean(y\*\*2)/2
37. j = abs((m4-2\*m2\*\*2)/(4\*Ps\*\*2))
38. **return** (r,j)
40. **def** feature\_Bispectrum(y):
41. ly = size  # 行数10
42. nrecs = np.int64(1 / sampling\_t)  # 列数100
43. nlag = 20
44. nsamp = nrecs  # 每段样本数100
45. nrecord = size
46. nfft = 128
47. Bspec = np.zeros((nfft, nfft), dtype=np.float32)
48. y = y.reshape(ly, nrecs)
49. c3 = np.zeros((nlag + 1, nlag + 1), dtype=np.float32)
50. ind = np.arange(nsamp)
52. **for** k **in** range(nrecord):
53. x = y[k][ind]
54. x = x - np.mean(x)
55. **for** j **in** range(nlag + 1):
56. z = np.multiply(x[np.arange(nsamp - j)], x[np.arange(j, nsamp)])
57. **for** i **in** range(j, nlag + 1):
58. sum = np.mat(z[np.arange(nsamp - i)]) \* np.mat(x[np.arange(i, nsamp)]).T
59. sum = sum / nsamp
60. c3[i][j] = c3[i][j] + sum  # i,j顺序
61. c3 = c3 / nrecord
63. c3 = c3 + np.mat(np.tril(c3, -1)).T  # 取对角线以下三角,c3为矩阵
64. c31 = c3[1:, 1:]
65. c32 = np.mat(np.zeros((nlag, nlag), dtype=np.float32))
66. c33 = np.mat(np.zeros((nlag, nlag), dtype=np.float32))  # 不可以直接3者相等
67. c34 = np.mat(np.zeros((nlag, nlag), dtype=np.float32))
68. **for** i **in** range(nlag):
69. x = c31[i:, i]
70. c32[nlag - 1 - i, 0:nlag - i] = x.T
71. c34[0:nlag - i, nlag - 1 - i] = x
72. **if** i < (nlag - 1):
73. x = np.flipud(x[1:, 0])  # 上下翻转,翻转后依然为矩阵
74. c33 = c33 + np.diag(np.array(x)[:, 0], i + 1) + np.diag(np.array(x)[:, 0], -(i + 1))
75. c33 = c33 + np.diag(np.array(c3)[0, :0:-1])
76. cmat = np.vstack((np.hstack((c33, c32, np.zeros((nlag, 1), dtype=np.float32))),
77. np.hstack((np.vstack((c34, np.zeros((1, nlag), dtype=np.float32))), c3))))          #41\*41
78. Bspec = fftpack.fft2(cmat, [nfft, nfft])
79. Bspec = np.fft.fftshift(Bspec)                #128\*128
81. waxis = np.arange(-nfft / 2, nfft / 2) / nfft
82. X, Y = np.meshgrid(waxis, waxis)
83. plt.contourf(X, Y, abs(Bspec),alpha=0,cmap=plt.cm.hot)
84. plt.contour(X, Y, abs(Bspec))
85. plt.show()
86. **return** Bspec
88. **def** features(s):
89. #    for mc in range(2):
90. snr = np.random.uniform(0, 20)       #从一个均匀分布集合中随机采样，左闭右开--[low,high)
91. s = awgn(s,snr)            #在原始信号的基础上增加SNR信噪比的噪音
92. rj = np.array(feature\_rj(s))               #计算R,J特征
93. z = feature\_Bispectrum(s)                  #计算双谱特征，并画图像
94. xx = np.int64(np.sqrt(np.size(z))/2)
95. z = np.array(z[:xx,xx:])
96. z = np.tril(z).real               #取复数z的实部
97. bis = np.zeros((1, xx),dtype=np.float32)    #零组
98. **for** i **in** range(xx):
99. **for** j **in** range(xx-i):
100. bis[0][i] = bis[0][i]+z[xx-1-j][i+j]
101. m = bis[0].reshape(1,xx)
102. normalized = preprocessing.normalize(m)[0,:]    #样本各个特征值除以各个特征值的平方之和
103. features = np.hstack((rj,normalized))           #合并数组r,j和normalized
104. **return** features
106. **print**(features(fsk))

好的，特征值已经有了，那么下面我们怎么使用这些特征值来描述初始信号的变化趋势呢？

画出特征值曲线是个不错的办法，趋势一目了然，为了说明信号受到影响，我们分别从信号的振幅A，角频率ω，初始频率S的改变来说明问题。初始信号图像如下：

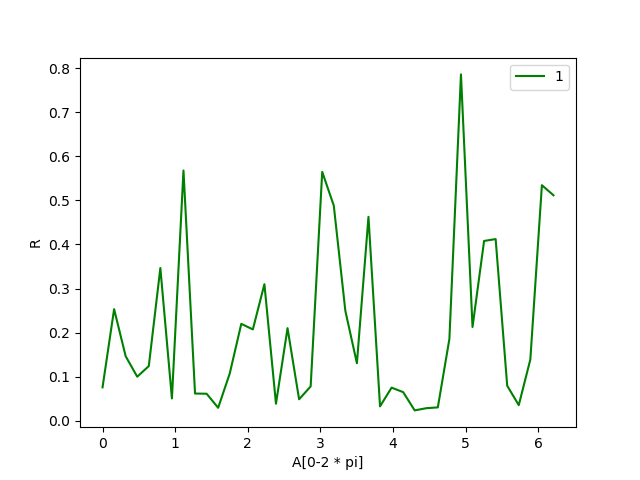


为了统一起见，我们分别另A，ω，S从0到2π均匀变化，对于振幅A的变化，补充代码计算R，J特征值，画出双谱图：

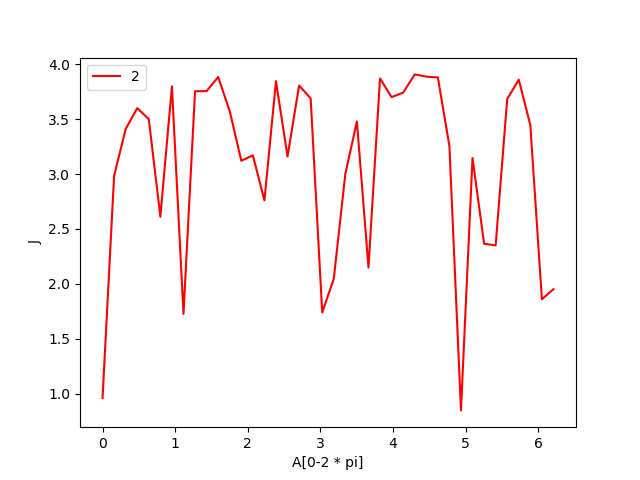
1. R = []
2. J = []
3. ts1 = np.arange(0, (np.int64(1/sampling\_t) \* size))/(10\*(m+1))
4. W = []
5. Z = []
6. **for** i **in** range(0,40,1):
7. W.append(i / (2 \* pi))
9. **for** a1 **in** W:
10. **global** r,j
11. fsk = a1 \* np.cos(np.dot(2 \* pi, ts1) + pi / 4)
12. features(fsk)
13. R.append(r)
14. J.append(j)
16. plt.plot(W, R, color='green', label='1')
17. plt.legend() # 显示图例
18. plt.xlabel('A[0-2 \* pi]')
19. plt.ylabel('R')
20. plt.show()
21. plt.plot(W, J, color='red', label='2')
22. plt.legend() # 显示图例
23. plt.xlabel('A[0-2 \* pi]')
24. plt.ylabel('J')
25. plt.show()

28. plt.plot(W, Z, color='red', label='3')
29. plt.legend() # 显示图例
30. plt.xlabel('A[0-2 \* pi]')
31. plt.ylabel('trend')
32. plt.show()

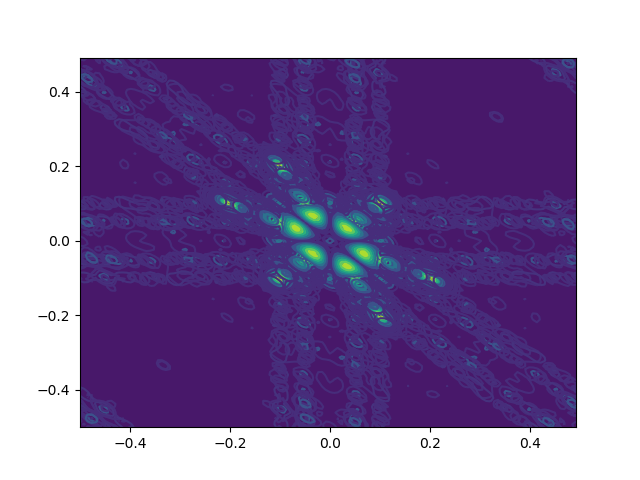
R特征值变化图像：



J特征值变化图像：



双谱图像：

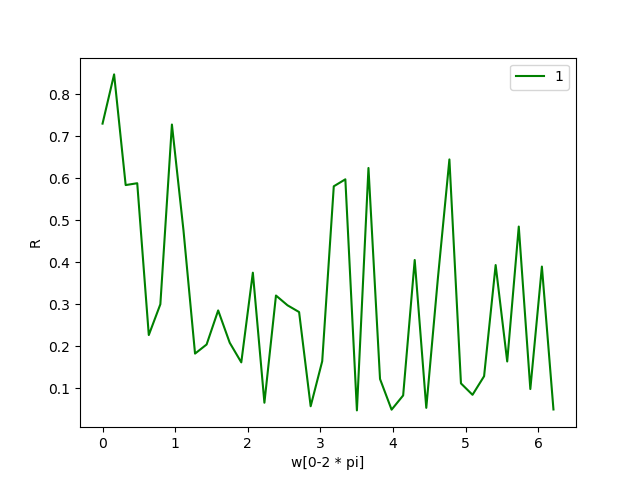


对于角频率ω的变化，补充代码计算R，J特征值，画出R、J、双谱图像：

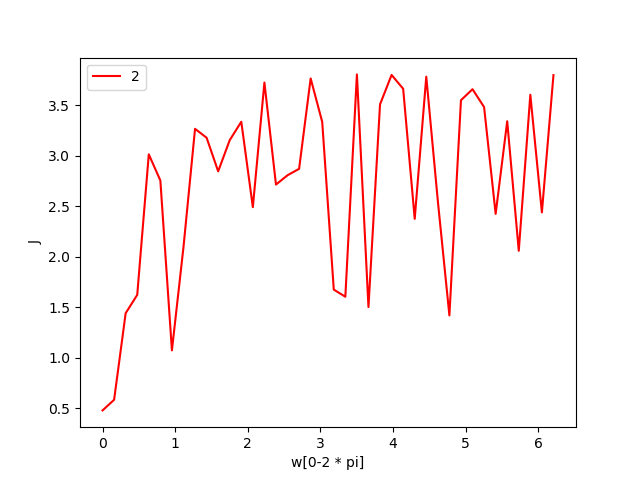
1. R = []
2. J = []
3. ts1 = np.arange(0, (np.int64(1/sampling\_t) \* size))/(10\*(m+1))
4. W = []
5. Z = []
6. **for** i **in** range(0,40,1):
7. W.append(i / (2 \* pi))
9. **for** w **in** W:
10. **global** r,j
11. fsk = np.cos(np.dot(w, ts1) + pi / 4)
12. features(fsk)
13. R.append(r)
14. J.append(j)
15. plt.plot(W, R, color='green', label='1')
16. plt.legend() # 显示图例
17. plt.xlabel('w[0-2 \* pi]')
18. plt.ylabel('R')
19. plt.show()
20. plt.plot(W, J, color='red', label='2')
21. plt.legend() # 显示图例
22. plt.xlabel('w[0-2 \* pi]')
23. plt.ylabel('J')
24. plt.show()

27. plt.plot(W, Z, color='red', label='3')
28. plt.legend() # 显示图例
29. plt.xlabel('A[0-2 \* pi]')
30. plt.ylabel('trend')
31. plt.show()

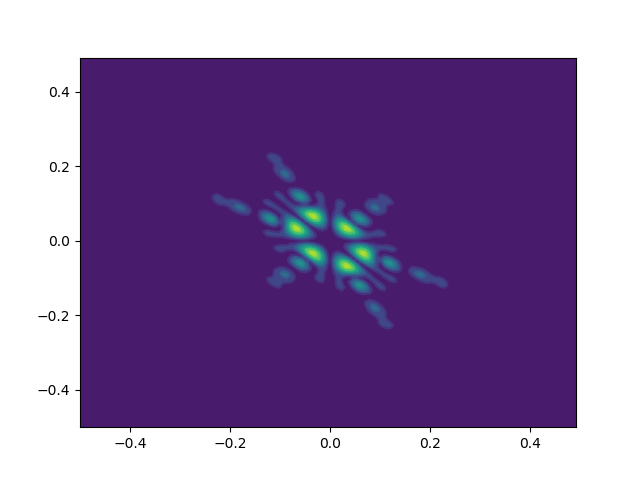
R特征值变化图像：



J特征值变化图像：



双谱图像：

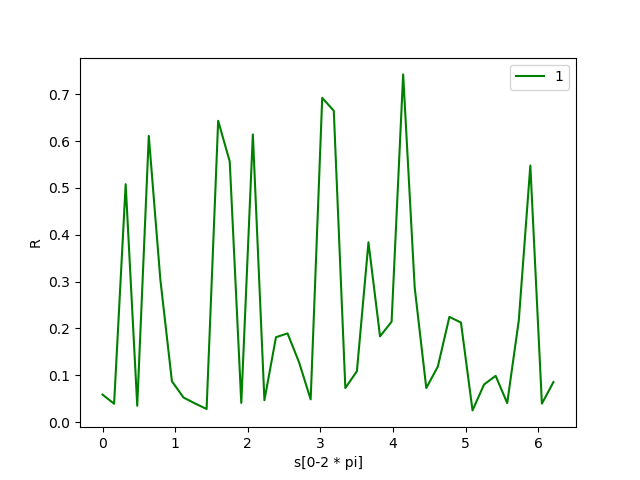


对于初始频率S的变化，补充代码计算R，J特征值，画出双谱图：

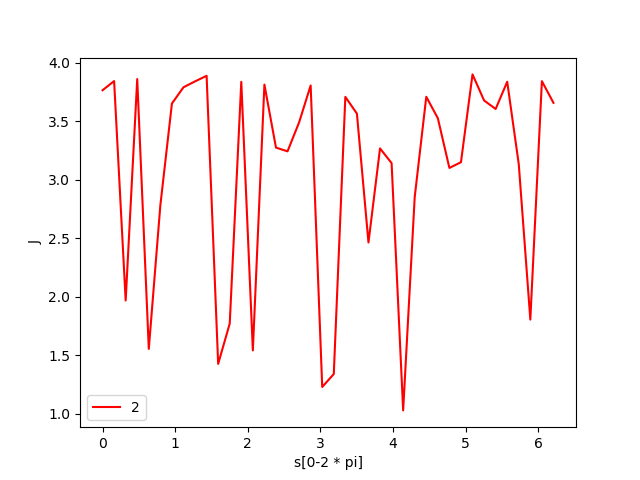
1. R = []
2. J = []
3. ts1 = np.arange(0, (np.int64(1/sampling\_t) \* size))/(10\*(m+1))
4. W = []
5. Z = []
6. **for** i **in** range(0,40,1):
7. W.append(i / (2 \* pi))
9. **for** s **in** W:
10. **global** r,j
11. fsk = np.cos(np.dot(2 \* pi, ts1) + s)
12. features(fsk)
13. R.append(r)
14. J.append(j)
15. plt.plot(W, R, color='green', label='1')
16. plt.legend() # 显示图例
17. plt.xlabel('s[0-2 \* pi]')
18. plt.ylabel('R')
19. plt.show()
20. plt.plot(W, J, color='red', label='2')
21. plt.legend() # 显示图例
22. plt.xlabel('s[0-2 \* pi]')
23. plt.ylabel('J')
24. plt.show()

27. plt.plot(W, Z, color='red', label='3')
28. plt.legend() # 显示图例
29. plt.xlabel('A[0-2 \* pi]')
30. plt.ylabel('trend')
31. plt.show()

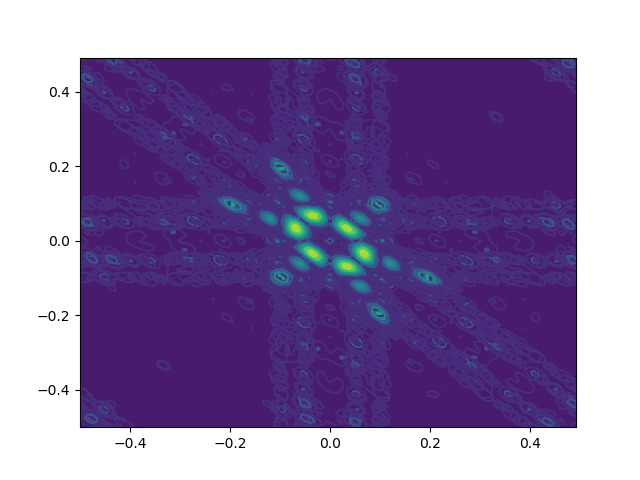
R特征值变化图像：



J特征值变化图像：



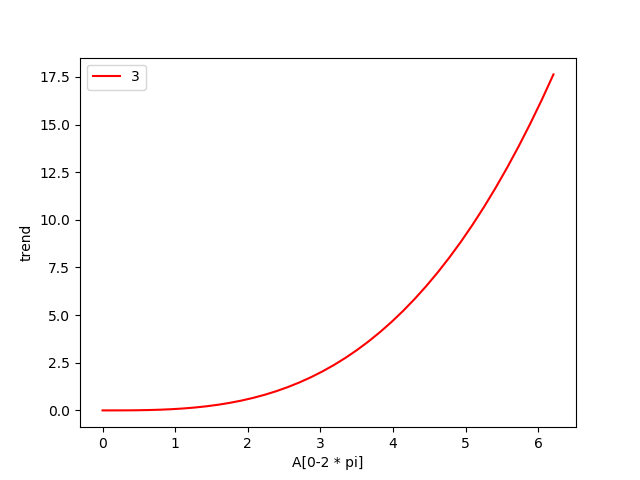
双谱图像：



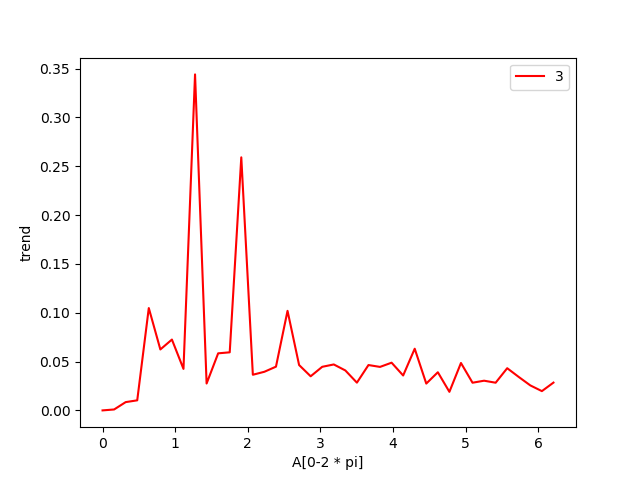
二维的双谱图像看起来不是很方便，那么对图像进行降维处理，像这种比较对称的矩阵，用矩阵平均值是最适合的了，这里只需将希尔伯特变换后的包络矩阵求下均值，改变下述代码即可：

1. Bspec = fftpack.fft2(cmat, [nfft, nfft])
2. Bspec = np.fft.fftshift(Bspec)                #128\*128
4. waxis = np.arange(-nfft / 2, nfft / 2) / nfft
5. X, Y = np.meshgrid(waxis, waxis)
6. plt.contourf(X, Y, abs(Bspec))
7. plt.contour(X, Y, abs(Bspec))
8. Z.append(np.mean(abs(Bspec)))
9. s\_content = './A幅度改变/'

则双谱特征降维后图像如下，振幅A的双谱特征变化情况：



角频率ω的双谱特征变化情况：



初始频率S的双谱特征变化情况：

