1 实验题目

- 基于 MATLAB 软件进行数字通信系统仿真。
- 培养学生综合运用《信号与系统》、《通信原理》等课程知识的能力；
- 培养学生模块化系统设计及系统开发的思想；
- 培养学生利用软件进行通信系统仿真的能力。
- 深入理解题目对应的通信原理基础知识，画出所仿真的通信系统**结构框图**及各个子模块的原理框图，写出相应模块输入、输出的数学表达式；
- 提出仿真方案，明确给出**仿真参数**；
- 完成 MATLAB 软件代码编制，重要的语句给出注释；主程序中至少包含一次自编**子程序的调用**。
- 采用**图形输出**演示设计结果；
- 撰写实验报告，**层次清晰**，结构完整；
- **附加项**：使用 MATLAB 自带 GUIDE 工具创建图形用户界面（Graphical User Interface, GUI）；
- **附加项**：利用 python 等其它高级语言；
- 装有 MATLAB 软件的计算机一台，选择适用版本。

1、基带码型仿真

1) 通过仿真观察占空比为 50%、75%以及 100%的单、双极性归零码波形以及其功率谱，分析不同占空比对仿真结果的影响，并总结单极性与双极性、不归零码和归零码的特点；

2) 通过仿真产生一**随机消息码**序列，将其分别转换为 AMI 码和 HDB_3 码，观察它们的波形，并说明与信源代码中的“1”码相对应的 AMI 码及 HDB3 码是否一定相同？给出下列源代码的 AMI 和 HDB3 码的代码和波形

0000 0000 0000 0000 0000 0000（全 0）

0100 0010 0000 1100 0011 0000

3) **附加题**：仿真观察 AMI 码和 HDB3 码的功率谱密度，并总结其特性；

2、数字带通调制仿真

1) 设计一个采用 2PSK 调制的数字通信系统：产生二进制随机数据，并仿真其对应的 2PSK 调制波形，观察并分析其频谱。所产生的调制波形加入不同信噪比的白噪声，选取合适的接收方案，画出系统误码率曲线，并与理论误码率进行对比。

2 实验原理

3 仿真方案

3.1 基于 Python 实现的方案

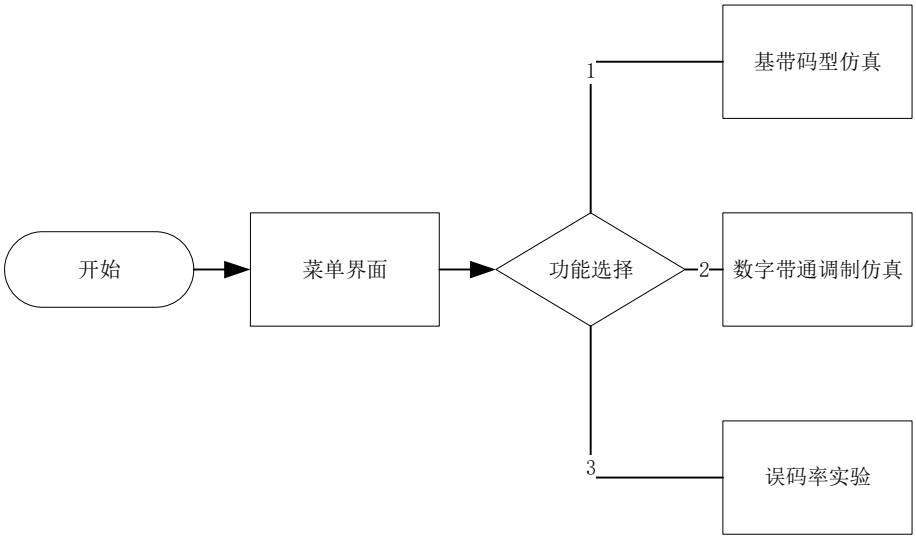

图 1 软件总体框图

总体上，多次使用自编库来实现功能。程序从 main.py 开始运行，调用了 bpskMod.py、conversionAMI.py、lowPassFilter.py、menuFunc.py、singleRZFunc.py、squareWave.py、whiteNoise.py 以及 errorRate.m 文件。除误码率计算使用了 python 中的 MATLAB engine 之外，其余均为 python 编程，使用了 matplotlib、NumPy 和 SciPy 等多个开源库。

3.2 基带码型仿真部分

在主菜单进入题目 1 后，再次选择仿真单双极性归零码或 AMI 与 HDB3 码转换仿真实验。

图 2 主菜单

3.2.1 单双极性归零码波形观察实验

功能的主体函数 RZFunc 在文件 singleRZFunc.py 之中。RZFunc 有一个输入参数，为 isSingle。主体逻辑框图如下所示：

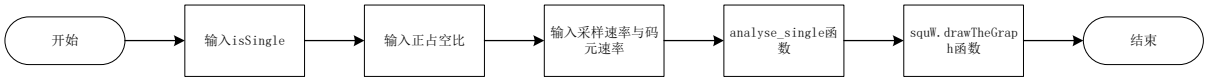

图 3 RZFunc 函数的流程图

其中，analyse_single 和 drawTheGraph 函数均来自 squareWave.py 文件。analyse_single 函数有五个参数，分别是 inputS（输入的序列）, duty（正占空比）, Fs（采样速率）, f（码元速率）, isSingle（默认为 1.isSingle==1 时按单极性归零码处理；反之则按照双极性归零码处理）。返回值有四个，分别是 output: List 输出的用来显示用的 List，纵轴；timeLine: List 输出的用来显示用的 List，横轴；fft_single: List 信号进行 fft 变换后的 List；f_single: List 频谱

的横轴。analyse_single 的作用是将波形进行采样以及 fft 变换。

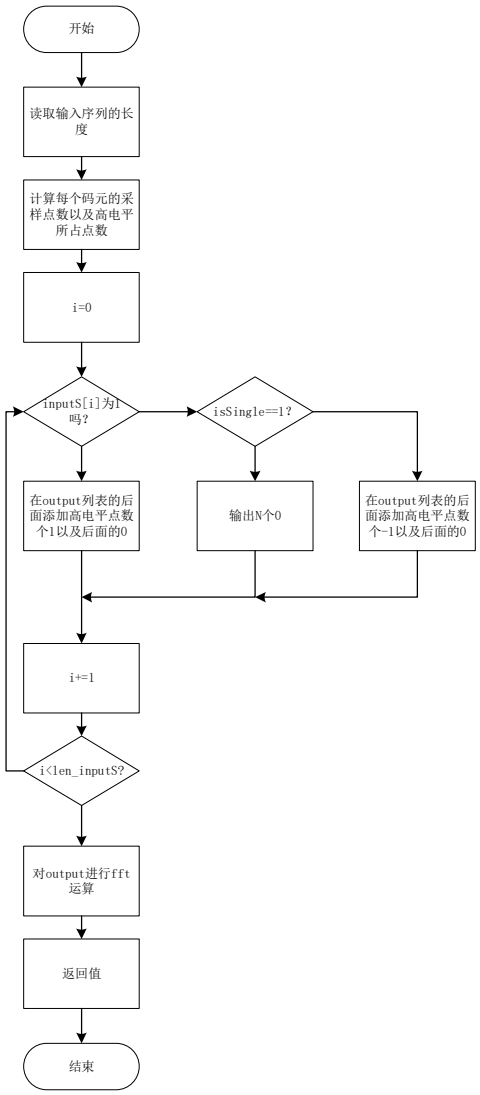

图 4 analyse_single **函数流程图**

drawTheGraph 函数的功能是将输入序列的波形、频谱、功率谱计算并显示出来。其中，功率谱的计算使用了 SciPy 科学计算库中的 periodogram 函数；画图使用了 matplotlib 中的 pyplot 库。

3.2.2 随机消息码序列转换为 AMI 与 HDB3 及其波形与功率谱实验

该功能的主要函数位于 conversionAMI.py 文件中。该函数名为 ami_hdb3show。

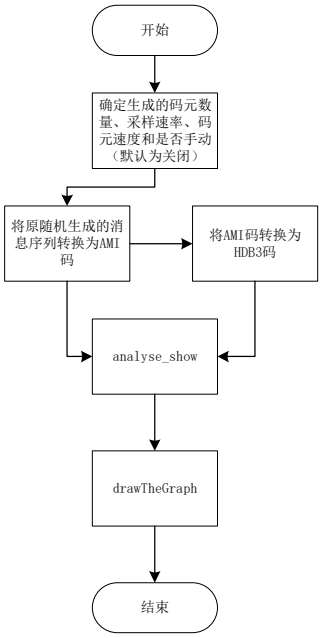

图 5 ami_hdb3show 的流程图

3.3 数字带通仿真实验

函数的主体为位于 bpskMod.py 文件中的 BPSK 函数。

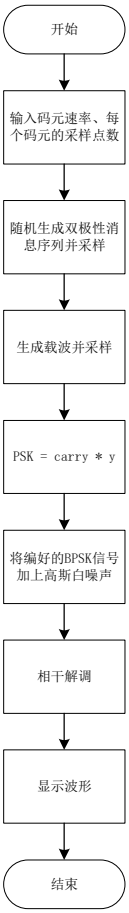

图 6 BPSK 函数流程图

将调制信号加入高斯白噪声时使用了 numpy.rand.randn 函数，生成符合正态分布的随机数，作为高斯白噪声混入调制后信号中。

其中相干解调时使用了巴特沃斯数字滤波器。函数均来自 scipy.signal 库中，filter_design.py 文件中。

3.4 误码率实验

利用了 MATLAB engine 库。但是由于电脑性能问题，如果运行计算 1E3 量级以上的白噪声->带通滤波变为窄带噪声->相干解调（乘载波、低通滤波）过程，计算机运行时长超过 7min 并报错（内存超过 1.44GiB）。

所以采用的误码率曲线测绘方法是直接向解调好的数字码元上加入窄带噪声后直接与原消息序列进行对比，再画为误码率曲线。

在 Python 中调用了 MATLAB engine。Python 3.8 对应的 MATLAB 版本为 R2020b。主要理由为要想实现误码率小于 1E-7 量级，就要使参与运算的码元达到 1E7 的量级。Python 中调制->加噪声->解调->比较这一流程，在量级为 1E3 时运行时长已经达到内存极限。所以选择了使用 MATLAB+窄带噪声作为运算误码率的仿真方案。

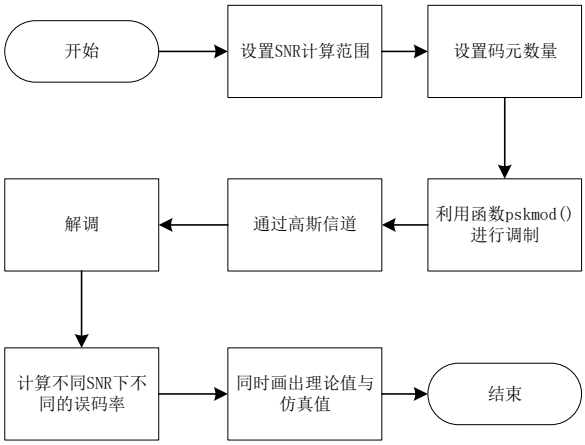

图 7 errorRate.m 的流程图

4 实验结果及分析

4.1 数字基带码型仿真实验结果

菜单选择上，再次选择功能 1。

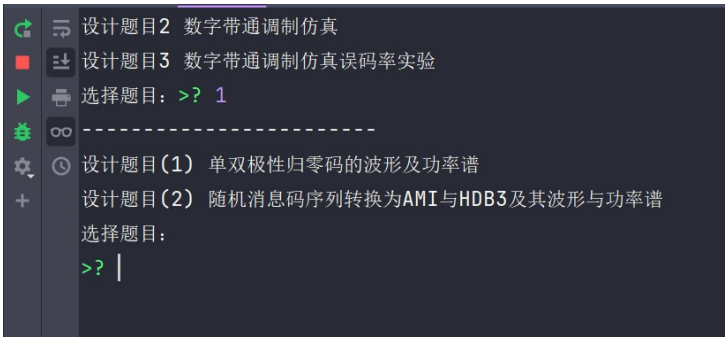

图 8 菜单 1-1

可以开始设置单双极性码的参数。以下图参数为例：

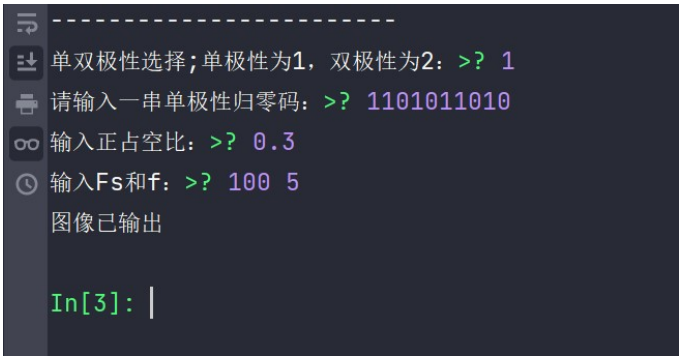

图 9 单极性归零码范例

可以得到仿真图表。

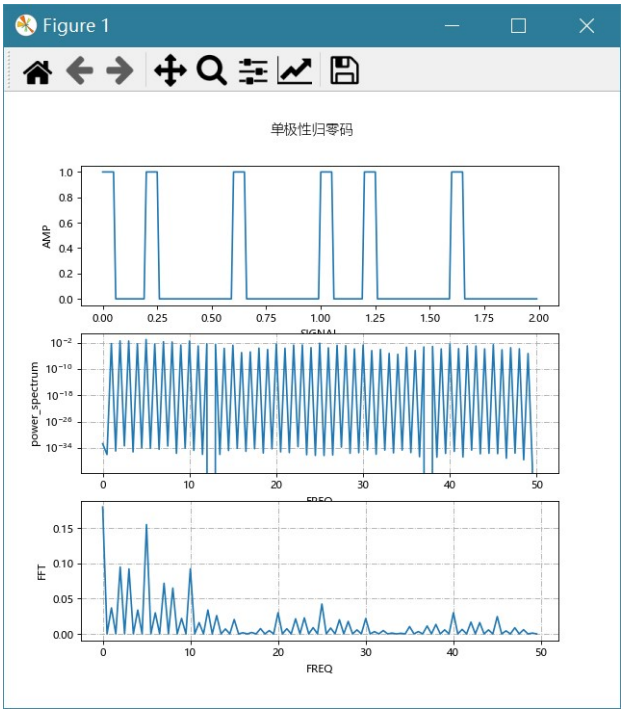

图 10 单极性归零码的波形与频谱图

再次选择题目 2，可进行随机消息码序列转换为 AMI 与 HDB3 及其波形与功率谱实验。

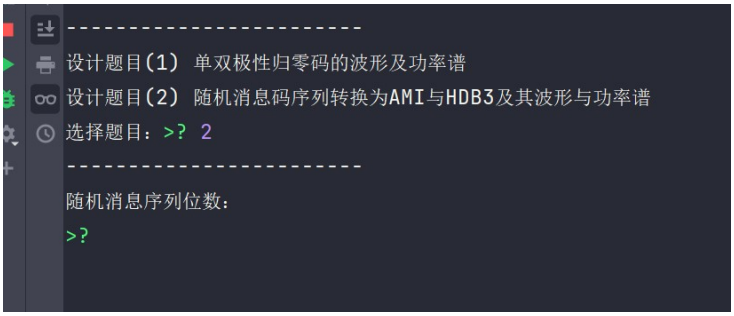

图 11 菜单 1-2

以下图的参数为例：

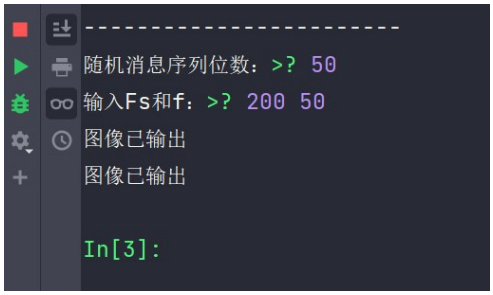

图 12 随机消息序列参数

得出波形图。

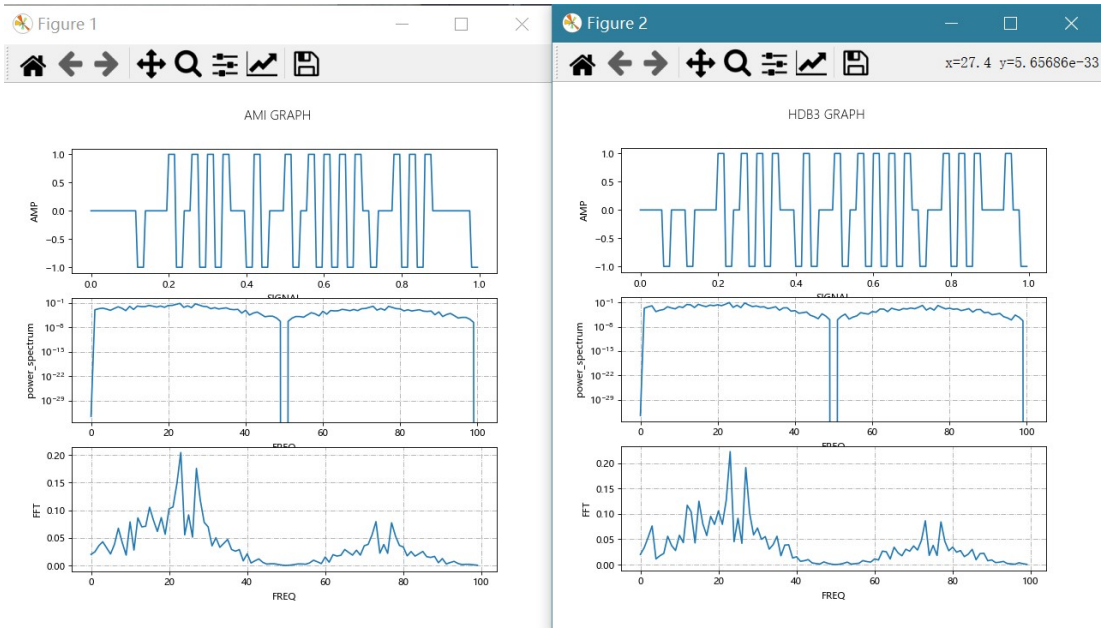

图 13 AMI 与 HDB3 码的波形、频谱与功率谱

4.2 数字带通仿真实验波形

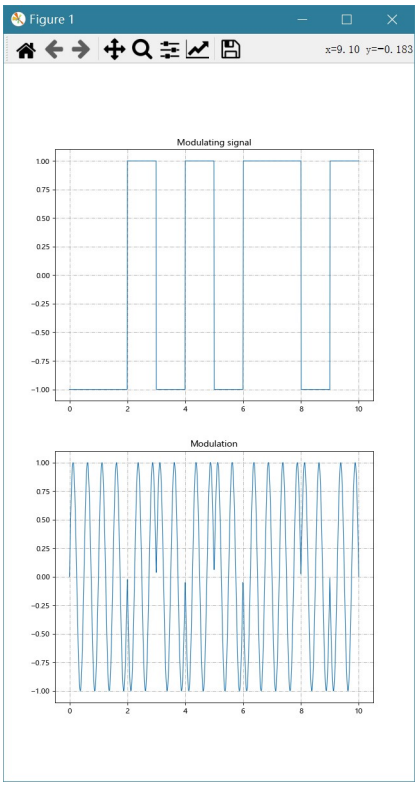

图 14 BPSK 调制波形

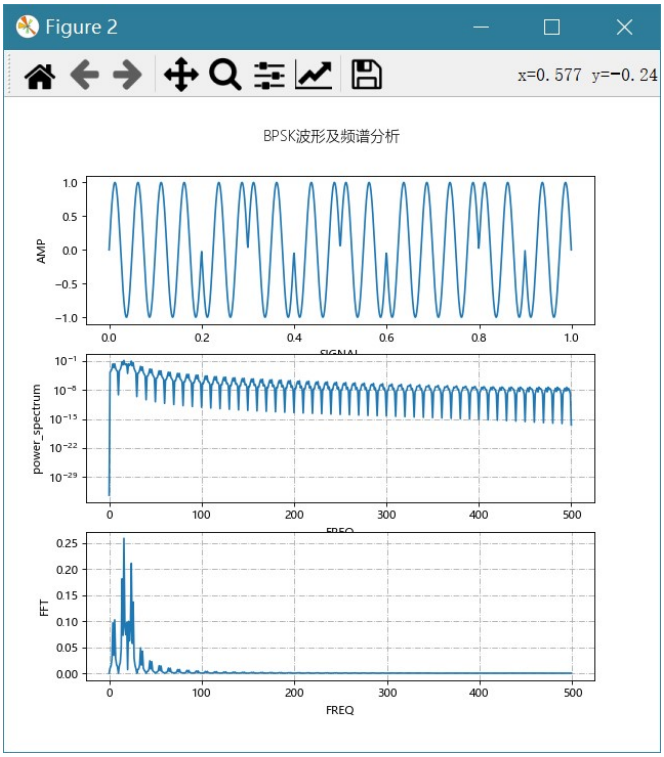

图 15 BPSK 的功率谱与频谱

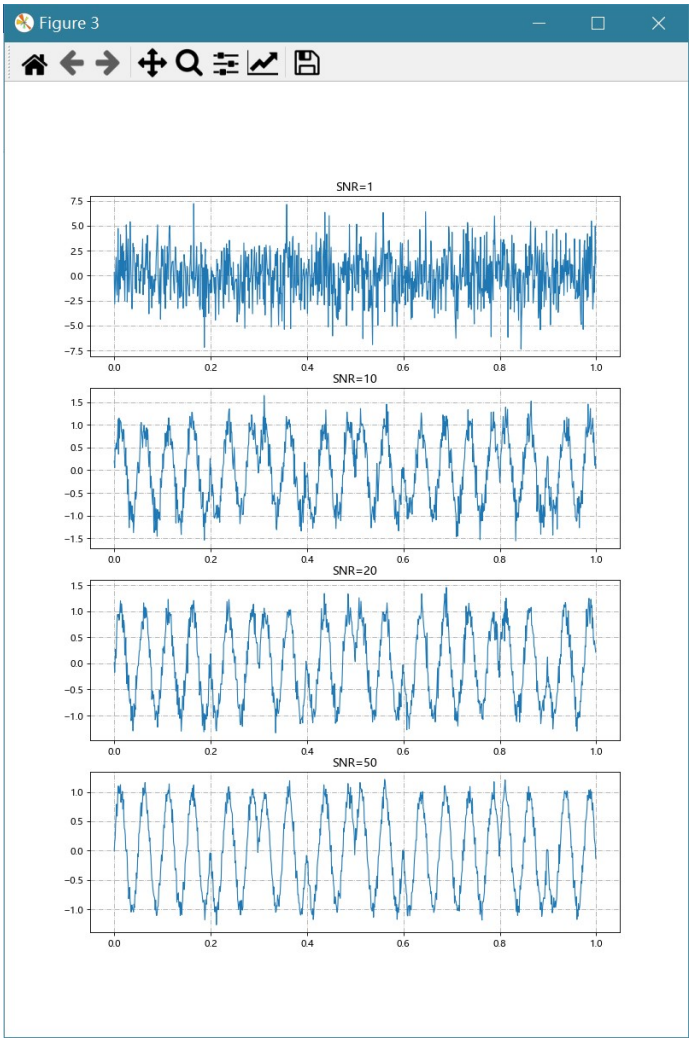

图 16 通过不同信噪比的高斯信道

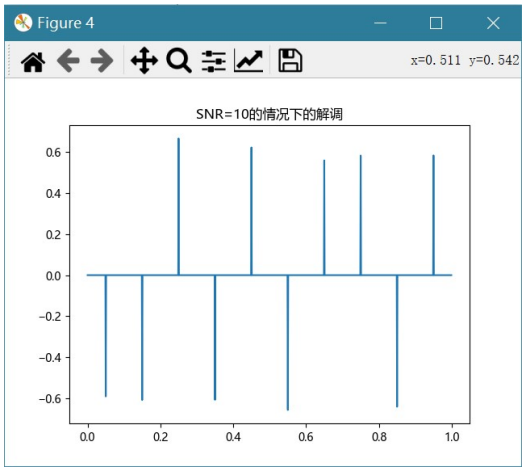

图 17 对相干解调波形抽样后得到的波形

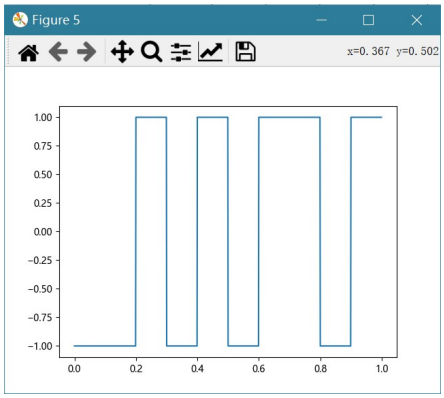

图 18 恢复的消息序列

4.3 误码率曲线

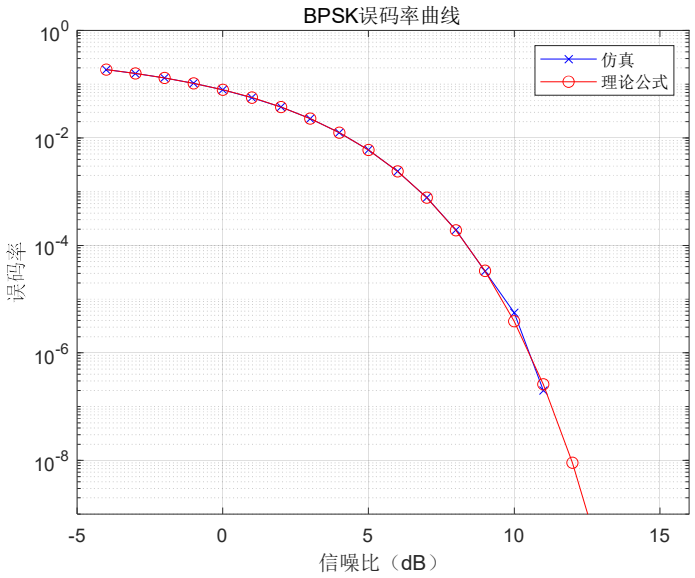

图 19 BPSK 的误码率曲线（仿真与理论公式对比）

4.4 实验结果分析

1. 不同占空比对仿真结果的影响

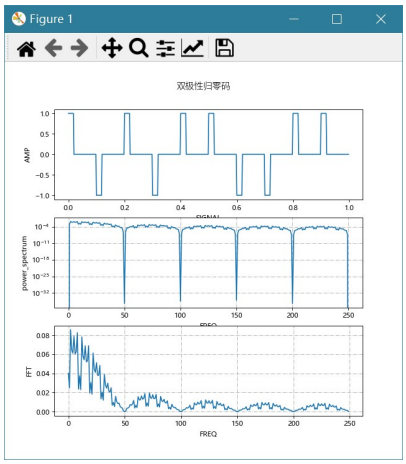

图 20 占空比 20%

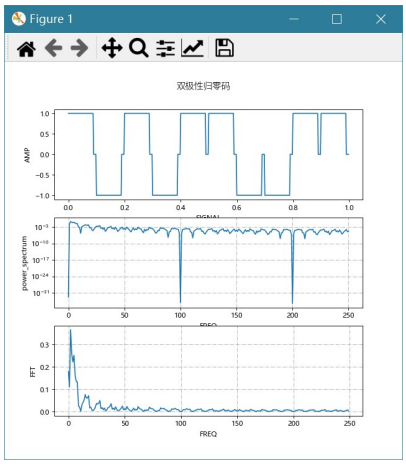

图 21 占空比 90%

占空比越大，频谱越集中。

5 总结

附录：程序源码

Main.py

```
import matplotlib

import singleRZFunc
import menuFunc as mf
import conversionAMI as ami
import bpskMod

matplotlib.rcParams['font.sans-serif'] = ['Microsoft YaHei']  # 设置字体为微软雅黑，以便输出图像中可以使用中文
q1 = mf.menu1()
if q1 == '1':  # 设计题目 1 基带码型仿真
    q1_1 = mf.menu1_1()
    if q1_1 == 1:  # 单双极性归零码
        choice = mf.menu1_1_1()
        singleRZFunc.RZFunc(choice)
    elif q1_1 == 2:  # AMI & HDB3
        number, Fs, f = mf.menu1_1_2()
        # ami.ami_show(number)
        ami.ami_hdb3show(number, Fs, f, 0)
elif q1 == '2':  # 设计题目 2 数字带通调制仿真
    bpskMod.BPSK()
elif q1 == '3':
    bpskMod.errorRate()
else:
    print("非法输入")
```

bpskMod.py

```
import matplotlib.pyplot as plt
import numpy as np
from squareWave import
drawTheGraph, fft_waveform
import whiteNoise as wN
import lowPassFilter as lpf
import matlab.engine

eng =
matlab.engine.start_matlab()


def BPSK():
    plt.figure(figsize=(8, 14),
dpi=80)
    plt.subplot(211)
    plt.title("Modulating
signal")
    print("仿真图窗 1s")
    f = int(input("请输入码元数(码
元速率)：："))
    N = int(input("请输入每个码元的
采样点数："))
    t = np.linspace(0, f, f * N)
# 时间轴
    y = np.ones((f, N), int)
    mt = []
    for i in range(f):
        r_code =
int(round(np.random.random()))
        y[i] = r_code * 2 - 1  # 双
极性
        mt.append(r_code * 2 - 1)
    y = y.flatten()  # 降维
    plt.grid(True,
linestyle='-.')
    plt.plot(t, y, linewidth=1,
linestyle="-")  # 绘制
    plt.locator_params(nbins=f)

    carrier = np.sin(1500 * np.pi
* t + np.pi)
    PSK = carrier * y
    plt.subplot(212)
    plt.title("Modulation")
    plt.grid(True,
linestyle='-.')
    plt.plot(t, PSK, linewidth=1,
linestyle="-")  # 绘制
    plt.show()

    timeLine, fft_single,
f_single = fft_waveform(PSK, f,
f * N, N)
    drawTheGraph(PSK, timeLine,
fft_single, f_single, f * N,
"BPSK 波形及频谱分析")

    signal_add_whiteNoise1 = PSK
+ wN.wgn(PSK, 0.1)
    plt.figure(figsize=(10, 14),
dpi=80)
    plt.subplot(411)
    plt.title("SNR=1")
    plt.grid(True,
linestyle='-.')
    plt.plot(timeLine,
signal_add_whiteNoise1,
linewidth=1, linestyle="-")  #
绘制

    signal_add_whiteNoise10 =
PSK + wN.wgn(PSK, 10)
    plt.subplot(412)
    plt.title("SNR=10")
    plt.grid(True,
linestyle='-.')
    plt.plot(timeLine,
signal_add_whiteNoise10,
linewidth=1, linestyle="-")  #
绘制

    signal_add_whiteNoise20 =
PSK + wN.wgn(PSK, 20)
    plt.subplot(413)
    plt.title("SNR=20")
    plt.grid(True,
linestyle='-.')
    plt.plot(timeLine,
signal_add_whiteNoise20,
```

```
linewidth=1, linestyle="-")  # 绘制

    signal_add_whiteNoise50 = PSK + wN.wgn(PSK, 50)
    plt.subplot(414)
    plt.title("SNR=50")
    plt.grid(True, linestyle='-.')
    plt.plot(timeLine, signal_add_whiteNoise50, linewidth=1, linestyle="-")  # 绘制
    plt.show()

    yt = (PSK + wN.wgn(PSK, 10)) * carrier
    y2, t = lpf.lpfDataOnly(2, N * f, f, yt)  # t 有 fs 个点，即 N*f
    position = int(N / 2)
    y_out = []
    for i in range(N * f):
        y_out.append(0)
    for i in range(f):
        y_out[i * N + position] = 1
    y2 = y_out * y2
    plt.figure()
    plt.title("SNR=10 的情况下的解调")
    plt.plot(t, y2)
    plt.show()

    demodulation_y = []
    y_demod = []
    for code in y2:
        if code == 0:
            continue
        elif code < 0:
            y_demod.append(-1)
            for i in range(N):

demodulation_y.append(-1)
        else:
            y_demod.append(1)
            for i in range(N):

demodulation_y.append(1)
    plt.figure()
    plt.plot(t, demodulation_y)
    plt.show()


def errorRate():
    eng.errorRate(nargout=0)
```

conversationAMI.py

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.signal import periodogram

from squareWave import analyse_show, drawTheGraph


def ori_ami(num=24, isManual=0):
    """
    :param isManual: 手动输入消息序列
    :param num: int, 产生消息序列长度
    :return: ami_signal:list,ami码
    """

    if isManual:
        # ori_signal = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        ori_signal = [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0]
        num = len(ori_signal)
    else:
        ori_signal = np.random.randint(0, 2, num)  # 产生 num 位随机二进制码
```

```
    am_signal = []
    # print("ORIGINAL:",
ori_signal)
    count_1 = 0
    # 转换为AMI码
    for i in range(0,
len(ori_signal)):
        if ori_signal[i] == 1:
            count_1 += 1
            if count_1 % 2 == 0:

am_signal.append(ori_signal[i])
            else:

am_signal.append(-ori_signal[i
])
        else:

am_signal.append(-ori_signal[i
])
    return am_signal


def ami_show(Fs, num=24,
isManual=0):
    amiList = ori_ami(num,
isManual)
    output, timeLine, fft_single,
f_single = analyse_show(amiList,
500, 10)
    drawTheGraph(output,
timeLine, fft_single, f_single,
Fs, "AMI GRAPH")
    return


def ami_hdb3(am_signal):
    count_0 = 0
    hdb3_signal = []
    for j in
range(len(am_signal)):
        if am_signal[j] == 0:
            count_0 += 1
            if count_0 == 4:

hdb3_signal.append(5)
                count_0 = 0
            else:

hdb3_signal.append(am_signal[j
])
        else:
            count_0 = 0

hdb3_signal.append(am_signal[j
])

    count_v = 0
    tmp = 0
    for k in
range(len(hdb3_signal)):
        if hdb3_signal[k] == 5:
            count_v += 1
            if count_v == 1 and
hdb3_signal[k - 4] is not None:
                hdb3_signal[k] =
hdb3_signal[k - 4]
                tmp =
hdb3_signal[k]
            else:
                if hdb3_signal[k -
4] == tmp:
                    hdb3_signal[k
- 3] = -tmp
                    hdb3_signal[k]
= hdb3_signal[k - 3]
                    tmp =
hdb3_signal[k]
                else:
                    hdb3_signal[k]
= hdb3_signal[k - 4]
                    tmp =
hdb3_signal[k]
    i = 0
    for code in hdb3_signal:
        if np.abs(code) == 5:
            hdb3_signal[i] =
int(code / 5)
        i += 1
    # print("HDB3:", hdb3_signal)
```

```
    return hdb3_signal


def ami_hdb3show(num=24, Fs=500,
f=10, isManual=0):
    amiList = ori_ami(num,
isManual)
    hdb3List = ami_hdb3(amiList)
    output, timeLine, fft_single,
f_single = analyse_show(amiList,
Fs, f)
    drawTheGraph(output,
timeLine, fft_single, f_single,
Fs, "AMI GRAPH")
    output, timeLine, fft_single,
f_single = analyse_show(hdb3List,
Fs, f)
    drawTheGraph(output,
timeLine, fft_single, f_single,
Fs, "HDB3 GRAPH")
    return


def powerSpec_Calculate():
    f_Pxx = []
    Pxx_remain_AMI = []
    Pxx_remain_HDB3 = []
    f = 100
    Fs = 1000
    N = int(Fs / f)
    times = 100
    for i in range(int(Fs / 2) +
1):
        Pxx_remain_AMI.append(0)
        Pxx_remain_HDB3.append(0)

    for calculate_time in
range(times):
        ami_random_signal =
ori_ami(num=f)
        hdb3_random_signal =
ami_hdb3(ami_random_signal)
        ami_random_signal_Fs = []
        hdb3_random_signal_Fs = []
        for i in ami_random_signal:
            for j in range(N):

ami_random_signal_Fs.append(i)

        for i in
hdb3_random_signal:
            for j in range(N):

hdb3_random_signal_Fs.append(i)

        f_Pxx, Pxx_AMI =
periodogram(ami_random_signal_
Fs, Fs)
        f_Pxx, Pxx_HDB3 =
periodogram(hdb3_random_signal
_Fs, Fs)
        for k in
range(len(Pxx_AMI)):
            Pxx_remain_AMI[k] +=
Pxx_AMI[k] / times
            Pxx_remain_HDB3[k] +=
Pxx_HDB3[k] / times
    plt.suptitle("功率谱密度",
fontsize=13, fontweight=0,
color='black', style='italic',
y=0.95)
    plt.subplot(211)
    plt.xlabel("AMI")
    plt.semilogy(f_Pxx,
Pxx_remain_AMI)
    plt.subplot(212)
    plt.xlabel("HDB3")
    plt.semilogy(f_Pxx,
Pxx_remain_HDB3)
```

lowpassFilter.py

```
import numpy as np
from matplotlib import pyplot as
plt
from scipy.signal import butter,
lfilter, freqz, filtfilt


def butter_lowpass(cutOff, fs,
```

```
order=5):
    """

    :param cutOff: 截至频率
    :param fs: 采样频率
    :param order: 阶数
    :return:
    """
    nyq = 0.5 * fs
    normalCutoff = cutOff / nyq
    b, a = butter(N=order,
Wn=normalCutoff)
    # butter(N, Wn, btype='low',
analog=False, output='ba',
fs=None)
    return b, a


def butter_bandpass(lowcut,
highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(N=order,
Wn=[low, high], btype='band')
    return b, a


def butter_lowpass_filter(data,
cutOff, fs, order=5):
    b, a = butter_lowpass(cutOff,
fs, order=order)
    # y = lfilter(b, a, data)   #
有时延的滤波
    y = filtfilt(b, a, data)  # 没
有时延的滤波
    return y


def butter_bandpass_filter(data,
lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut,
highcut, fs, order=order)
    y = filtfilt(b, a, data)
    return y
```

```
def lpfDataOnly(order, fs,
cutoff, data):
    """
    应用低通滤波器后的波形输出。
    :param order: filter 的阶数
    :param fs:采样频率
    :param cutoff:截止频率
    :param data:要过滤的波形数据
    :return: y,t
    """

    t = np.linspace(0, 1, fs,
endpoint=False)  # "Noisy" data.
We want to recover the 1.2 Hz
signal from this.
    y =
butter_lowpass_filter(data,
cutoff, fs, order)

    return y, t


def bpfDataOnly(order, fs,
cutoff1, cutoff2, data):
    """
    应用低通滤波器后的波形输出。
    :param order: filter 的阶数
    :param fs:采样频率
    :param cutoff1:截止频率 1
    :param cutoff2:截止频率 2
    :param data:要过滤的波形数据
    :return: y,t
    """

    t = np.linspace(0, 1, fs,
endpoint=False)
    y =
butter_bandpass_filter(data,
cutoff1, cutoff2, fs, order)

    return y, t
```

menuFunc.py

```
def menu1():
    print("------通信原理软件实验------")
    print("设计题目 1 基带码型仿真")
    print("设计题目 2 数字带通调制仿真")
    print("设计题目 3 数字带通调制仿真误码率实验")
    # print("设计题目 4 附加题：AMI 与 HDB3 的功率谱密度")
    q1 = input("选择题目：")
    return q1


def menu1_1():

print("------------------------")
    print("设计题目(1) 单双极性归零码的波形及功率谱")
    print("设计题目(2) 随机消息码序列转换为 AMI 与 HDB3 及其波形与功率谱")
    q1_1 = int(input("选择题目："))
    return q1_1


def menu1_1_1():

print("------------------------")
    answer = int(input("单双极性选择;单极性为 1，双极性为 2："))
    if answer == 1:
        q1_1_1 = True
    elif answer == 2:
        q1_1_1 = False
    else:
        print("input error")
        return
    return q1_1_1


def menu1_1_2():

print("------------------------")
    answer = int(input("随机消息序列位数："))
    Fs, f = input("输入 Fs 和 f：").split()
    Fs = int(Fs)
    f = int(f)
    return answer, Fs, f
```

singleRZFunc.py

```
import squareWave as squW


def RZFunc(isSingle=1):
    """
    Parameters
    ----------
    isSingle: bool
        是单极性，isSingle=1；是双极性，isSingle=0
        默认为 1
    """
    if isSingle:
        single = input("请输入一串单极性归零码：")
        title = "单极性归零码"
    else:
        single = input("请输入一串双极性归零码：")
        title = "双极性归零码"
    duty = float(input("输入正占空比："))
    Fs, f = input("输入 Fs 和 f：").split()
    Fs = int(Fs)
    f = int(f)
    wave_single, timeLine, fft_single, f_single = squW.analyse_single(single, duty, Fs, f, isSingle)

squW.drawTheGraph(wave_single, timeLine, fft_single, f_single,
```

```
Fs, title)
    return
```

squareWave.py

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.signal import periodogram


def fft_waveform(waveform, length, Fs, N):
    """
    :param waveform: list,需要处理的波形
    :param length: int,信息序列长度
    :param Fs: 采样速率
    :param N: 采样点数/码元
    :return: timeLine, fft_single, f_single
    """
    # timeLine = []
    t = length * N / Fs
    timeLine = np.arange(0, t, 1 / Fs)
    num_fft = len(waveform)
    fft_single = np.abs(np.fft.fft(waveform, num_fft))  # 进行傅里叶变换，前者为数据，后者为数据长度
    f_single = np.arange(num_fft)[range(int(len(timeLine) / 2))] / t
    # fft_single = np.abs(fft_single[1:1 + int((len(waveform)) / 2)]) / num_fft
    fft_single = np.abs(fft_single[range(int(len(fft_single) / 2))]) / num_fft
    return timeLine, fft_single, f_single


def analyse_single(inputS, duty, Fs, f, isSingle=1):
    """
    Parameters
    ----------
    inputS: str
        输入的单or双极性归零码
    duty: float
        正占空比
    Fs: int
        采样速率
    f: int
        码元速率
    isSingle: bool
        是单极性，isSingle=1；是双极性，isSingle=0
        默认为1

    Returns
    -------
    output: List
        输出的用来显示用的List，纵轴
    timeLine: List
        输出的用来显示用的List，横轴
    fft_single: List
        信号进行fft变换后的List
    f_single: List
        频谱的横轴
    """
    output = []
    len_inputS = len(inputS)  # 读取输入序列的长度
    N = int(Fs / f)  # 每个码元的采样点数
    N_high = int(duty * N)  # 高电平所占点数
    for i in range(0, len_inputS):
        if inputS[i] == '1':
            for j in range(i * N, i * N + N_high):  # 码元开始时开始到duty%结束
                output.append(1)
```

```
            # 输出 1
            for j in range(i * N +
N_high, (i + 1) * N):
                output.append(0)
            # 输出 0
        else:
            if isSingle:
                for j in range(i *
N, (i + 1) * N):

output.append(0)
            else:
                for j in range(i *
N, i * N + N_high):

output.append(-1)
                for j in range(i *
N + N_high, (i + 1) * N):

output.append(0)  # 输出 0

    timeLine, fft_single,
f_single = fft_waveform(output,
len_inputS, Fs, N)
    return output, timeLine,
fft_single, f_single


def analyse_show(listIn, Fs, f):
    """
    :param listIn: 需要显示出波形的
list 输入
    :param Fs: 采样频率
    :param f: 码元速率
    :return:output, timeLine,
fft_single, f_single

    """
    output = []
    len_listIn = len(listIn)
    N = int(Fs / f)
    i = 0
    for code in listIn:
        for j in range(i * N, (i +
1) * N):
            output.append(code)
        i += 1
    timeLine, fft_single,
f_single = fft_waveform(output,
len_listIn, Fs, N)
    return output, timeLine,
fft_single, f_single


def drawTheGraph(wave_single,
timeLine, fft_single, f_single,
Fs, title="Waveform Graph"):
    """
    画图函数
    :param wave_single: 用来显示用
的 List，纵轴
    :param timeLine: 用来显示用的
List，横轴
    :param fft_single: 信号进行
fft 变换后的 List
    :param f_single: 频谱的横轴
    :param Fs: 采样率
    :param title: graph's title
    :return: N/A
    """
    f_Pxx, Pxx =
periodogram(wave_single, Fs)
    plt.figure(figsize=(8, 8),
dpi=80)
    plt.suptitle(title,
fontsize=13, fontweight=0,
color='black', style='italic',
y=0.95)
    # 两幅图像分开输出
    plt.subplot(311)
    plt.xlabel("SIGNAL")
    plt.ylabel("AMP")
    plt.plot(timeLine,
wave_single)
    plt.subplot(312)
    plt.grid(True,
linestyle='-.')
    plt.xlabel("FREQ")
    plt.ylabel("power_spectrum")
    plt.semilogy(f_Pxx, Pxx)
```

```
    plt.subplot(313)
    plt.grid(True,
linestyle='-.')
    plt.xlabel("FREQ")
    plt.ylabel("FFT")
    plt.plot(f_single,
fft_single)
    plt.show()
    print("图像已输出")
    return
```

whiteNoise.py

```
import numpy as np


def wgn(x, snr):
    Ps = np.sum(abs(x)**2)/len(x)
    Pn = Ps/(10 ** (snr / 10))
    noise =
np.random.randn(len(x)) *
np.sqrt(Pn)
    return noise
```

errorRate.m

```
function errorRate()
    clear
    clc

    SNR=-4:1:15;
    snr=10.^(SNR/10);
    N=10^7;
    M=2;
    x=randi([0,1],1,N);
;
    y=pskmod(x,M);

    for i=1:length(SNR)
        N0=1/2/snr(i);
        N0_dB=10*log10(N0);
        ni=wgn(1,N,N0_dB);

        yAn=y+ni;
        yA=pskdemod(yAn,M);

bit_A=length(find(x~=yA));
        BPSK_s_AWGN(i)=bit_A/N;

    end

BPSK_t_AWGN=1/2*erfc(sqrt(snr)
);%AWGN 信道下 BPSK 理论误码率

    %绘制图形
    figure;

semilogy(SNR,BPSK_s_AWGN,'-bx'
);hold on;

semilogy(SNR,BPSK_t_AWGN,'-ro'
);hold on;
    axis([-5,16,10^-9,1]);
    grid on
    legend('仿真','理论公式');
    title('BPSK 误码率曲线');
    xlabel('信噪比（dB）');
    ylabel('误码率')
```