writereadbits

February 9, 2017

1 Program - writereadbits

Module for writing to and reading binary file respectively. User Writes the the variable using appropriate function for writing and then can read it using the function for reading it. Below are the function definitions.

```
In [1]: print "writereadbits.py"
```

```
writereadbits.py
```

- Function for writing the bits to a fileUsage:

```
writebitsfile('filename.bin', bitstring)
```

  writes bitstring into binary file 'filename.bin'

```
In [2]: def writebinaryfile(filename, bitstring):
            import numpy as np
            import struct

            #use unsigned Bytes ('B'),
            #converts stream of bits into stream of Bytes (groups of 8) represented as decimal i
            numbytes=len(bitstring)/8;
            Bytes=np.zeros(numbytes,dtype=int)
            for m in range(numbytes):
               Bytes[m]=eval('0b'+bitstring[m*8:(m*8+8)])
            #packs the Bytes into a stream:
            s=struct.pack('B'*len(Bytes), *Bytes)
            #Write the stream "s" to file:
            file=open(filename, "w")
            #'w' deletes exisiting file, 'a' appends
            file.write(s)
            file.close()
```

- Function for reading from a binary file usage:

```
bitstring=readbinaryfile('filename.bin')
```

reads binary data from file 'filename.bin' and converts it into a string of bits.

```
In [3]: def readbinaryfile(filename):

            import struct
            import numpy as np

            file=open(filename, "r")
            #read the stream from file:
            readdata=file.read()

            #unpacks the stream into an array of Bytes (8 bits):
            Bytesread=struct.unpack('B'*len(readdata),readdata)

            #Convert bytes into binary string:
            bitstring='';
            for byte in Bytesread:
                #create bit strin from byte:
                bits=bin(byte)

                #remove leading '0b' and fill up to 8 bits with leading zeros:
                bits=bits[2:].zfill(8)

                #append to bits to bitstring:
                bitstring=bitstring+bits

            return bitstring
```

- Encoder function: data to codestring usage:

```
codestring = data2codestring(data)
```

takes 2 bit values in array "data" and returns the string of codewords

```
In [4]: def data2codestring(data):
            #usage: codestring=data2codestring( data)
            #takes 2 bit values in array "data" and returns the string of codewords

            import struct
            import numpy as np

            #sequence of integers (indices):
            #limit data to range of codebook (-2 to 1):
            data=np.clip(data,-2,1)

            #binary codebook, turns indices into binary codewords:
            #also works for negative numbers
            #(sog. Zweierkomplement):
```

```
        #https://de.wikipedia.org/wiki/Zweierkomplement
        #fuer negative Zahlen invertiere alle bits und addiere 1:
        codeword={0:'00',1:'01',-2:'10',-1:'11'}

        #start with empty string:
        codestring='';
        for value in data:
           #append new codewords:
           codestring=codestring+codeword[value]
        #print bits

        return codestring
```

- Decoder function usage:

```
data=codestring2data(codestring)
```

reads binary data from codestring and converts it into a stream of 2 bit values returned in data.

```
In [5]: def codestring2data(codestring):

            import struct
            import numpy as np

            #Bits decoder, decodes the indices from the binary codewords:
            decodeword = {'00':0,'01':1,'10':-2,'11':-1}

            #convert sequence of Bytes into sequence of bits:
            numdata = len(codestring)/2;

            #converts groups of 2 bits into data:
            data = np.zeros(numdata,dtype=int)
            n=0
            for i in range(numdata):
                #print "i= ", i
                data[n]=decodeword[codestring[(i*2):(2+i*2)]]
                #print "data[n]= ", data[n]
                n+=1
            return data
```

- Testing the module for storing an array, then reading it, then coverting it to codestring and finally decoding it back.

```
In [6]: if __name__ == '__main__':
            import numpy as np

            #emulation of an encoder, example data:
```

```
            data=np.array([-2,-1,0,1,-2,-1,0,1])

            print "data: ", data

            filename='savebin.bin'

            codestring=data2codestring(data)

            print "codestring: ", codestring

            #write to binary file
            writebinaryfile(filename, codestring)

            #Emulation of a decoder:
            codestring=readbinaryfile(filename)
            dataread=codestring2data(codestring)

            print "dataread: ", dataread
```

```
data:  [-2 -1  0  1 -2 -1  0  1]
codestring:  1011000110110001
dataread:  [-2 -1  0  1 -2 -1  0  1]
```