Supplementary Data 2: Readme file

The code in this directory demonstrates the Hamming coding/decoding for pyrosequencing reads, and the unit tests for this code.

To run the unit tests, type at the command-line:

```
python test_standalone_hamming.py
```

All tests should run without failure.

To run the demo, type at the command-line:

```
python standalone_hamming.py
```

You should get results identical to those in the file demo_output.txt.

Python and the Numpy extension module are required.
Supplementary Data 2: Decoding software example

Standalone Hamming(5, 11) demo decoder for 8 nt codewords (16 bits)

Requires python and numpy

Run demo via command line: "python standalone_hamming.py"

Author: Micah Hamady (hamady@colorado.edu)

from numpy import array

# current encoding scheme
INT_TO_BS = {0:"00", 1:"01", 2:"10", 3:"11"}
CUR_ENC_FO = {'A': 3, 'C': 2, 'T': 0, 'G': 1}
CUR_REV_ENC_SI = {"11":"A","10":"C","00":"T","01":"G"}

def calc_parity_vector(parity_vector):
    """ Returns even or odd parit for parity vector """
    return reduce(lambda x, y: x^y, parity_vector[1:])

def calc_syndrome(codeword, n):
    """ Calculate syndrome and correct codeword if possible """
    sym = 0
    for i in range(1,n):
        if codeword[i]:
            sym ^= i
    extra_parity = calc_parity_vector(codeword)
    if extra_parity == codeword[0]:
        if sym == 0:
            return 0, sym
        else:
            return 2, sym
    else:
        if sym >= n:
            pass
        else:
            codeword[sym] ^= 1
            return 1, sym

def nt_to_cw(cur_enc, cur_nt):
    """ Convert nt sequence to codeword """
    return array(map(int, ''.join([INT_TO_BS[cur_enc[x]] for x in cur_nt])))
def unpack_bitstr(rev_cur_bit, bitstr):
    """ Unpack bistring into nt sequence """
    bstr_len = len(bitstr)
    return ''.join([rev_cur_bit[bitstr[i:i+2]] for i in range(0, bstr_len, 2)])

def decode_barcode_8(nt_barcode):
    """ Decode length 8 barcode (16 bits) """
    # check proper length
    if len(nt_barcode) != 8:
        raise ValueError, "barcode must be 8 nt long."

    # check valid characters
    if set(list(nt_barcode)).difference(CUR_ENC_FO.keys()):
        raise ValueError, "Only A,T,C,G valid chars."

    # decode
    decoded = nt_to_cw(CUR_ENC_FO, nt_barcode)
    num_errors, sym = calc_syndrome(decoded, 16)

    # check errors
    if num_errors > 1:
        raise ValueError, "2 bit error detected."

    # convert corrected codeword back to nt sequence
    if num_errors == 1:
        nt_barcode = unpack_bitstr(CUR_REV_ENC_SI, ''.join(map(str, decoded)))

    return nt_barcode

# mapping from barcode used to original sample id
DEMO_SAMPLE_MAPPING = {
    "AACCATGC":"Sample 1",
    "TCGTAGCA":"Sample 2",
    "ACACCTCT":"Sample 3",
    "CTTCCTAG":"Sample 4",
    "GGTAGCTT":"Sample 5"
}

# dummy barcode tagged sequences (e.g. from 454 run)
DEMO_SEQUENCE = [
    # these all have no bit errors
    "AACCATGCTTTTTTTTTTTTTTTTTTTTTTTTT_0",
    "TCGTAGCATTTTTTTTTTTTTTTTTTTTTTTTT_1",
]
def demo():
    """ Demo decoding a list of tagged reads from several samples """
    print """"---------------------------------------"
    print "Processing %d sequences from %d samples" % (
        len(DEMO_SEQUENCES), len(DEMO_SAMPLE_MAPPING))
    print """"---------------------------------------"

    for ix, cur_seq in enumerate(DEMO_SEQUENCES):
        barcode = cur_seq[:8]
        seq_read = cur_seq[8:]
        print "--> processing demo sequence", ix
        print "read barcode     ", barcode
        try:
            corrected_barcode = decode_barcode_8(barcode)
            orig_sample_id = DEMO_SAMPLE_MAPPING[corrected_barcode]
if corrected_barcode != barcode:
    print "*corrected barcode:", corrected_barcode
else:
    print "-no error barcode:", corrected_barcode

print "original sample id:", orig_sample_id
print "sequence read :", seq_read

except ValueError, e:
    print "!", str(e), "skipping..."
    continue

if __name__ == "__main__":
    from sys import argv, exit

    print "Running demo..."
    demo()
    print "Demo done."
Supplementary Data 2: Decoding software example unit tests

Tests for standalone decoding function

Author: Micah Hamady (hamady@colorado.edu)

from unittest import TestCase, main
from standalone_hamming import *

class GeneralSetUp(TestCase):

def setUp(self):
    """General Setup"""
    # valid code words
    self.valid_bc_1 = "AACCATGC"
    self.valid_bc_2 = "TCGTAGCA"
    self.valid_bc_3 = "ACACCTCT"
    self.valid_bc_4 = "CTTCCTAG"
    self.valid_bc_5 = "GGTAGCTT"

    # single error reference
    self.single_error_ref = "AACCATGC"

    # A->C
    self.single_error_1 = "ACCCATGC"

    # A->G
    self.single_error_2 = "AACCGTGC"

    # C->A
    self.single_error_3 = "AAACATGC"

    # C->T
    self.single_error_4 = "AACCATGT"

    # T->C
    self.single_error_5 = "AACCACGC"

    # T->G
    self.single_error_6 = "AACCAGGC"

    # G->T
self.single_error_7 = "AACCATTC"

# G->A
self.single_error_8 = "AACCATGC"

# double error reference
self.double_error_ref = "AACCATGC"

# A->T
self.double_error_1 = "ATCCATGC"

# T->A
self.double_error_2 = "AACCAAGC"

# C->G
self.double_error_3 = "AACGATGC"

# aG->C
self.double_error_4 = "AACCATCC"

class StandaloneHammingTests(GeneralSetUp):
    """Tests for the standalone_hamming functions"""

def test_decode_barcode_8_ok(self):
    """Should decode valid codewords w/o error """
    self.assertEqual(decode_barcode_8(self.valid_bc_1),
                     self.valid_bc_1)
    self.assertEqual(decode_barcode_8(self.valid_bc_2),
                     self.valid_bc_2)
    self.assertEqual(decode_barcode_8(self.valid_bc_3),
                     self.valid_bc_3)
    self.assertEqual(decode_barcode_8(self.valid_bc_4),
                     self.valid_bc_4)
    self.assertEqual(decode_barcode_8(self.valid_bc_5),
                     self.valid_bc_5)

def test_decode_barcode_8_one_error(self):
    """Should correct single bit errors w/o error """
    self.assertEqual(decode_barcode_8(self.single_error_1),
                     self.single_error_ref)
    self.assertEqual(decode_barcode_8(self.single_error_2),
                     self.single_error_ref)
    self.assertEqual(decode_barcode_8(self.single_error_3),
                     self.single_error_ref)
from unittest import TestCase

class TestBarcodeDecoder(TestCase):

    def test_decode_barcode_8(self):
        # Should work if there are no errors
        self.assertEqual(decode_barcode_8(self.single_error_1),
                         self.single_error_ref)
        self.assertEqual(decode_barcode_8(self.single_error_2),
                         self.single_error_ref)
        self.assertEqual(decode_barcode_8(self.single_error_3),
                         self.single_error_ref)
        self.assertEqual(decode_barcode_8(self.single_error_4),
                         self.single_error_ref)
        self.assertEqual(decode_barcode_8(self.single_error_5),
                         self.single_error_ref)
        self.assertEqual(decode_barcode_8(self.single_error_6),
                         self.single_error_ref)
        self.assertEqual(decode_barcode_8(self.single_error_7),
                         self.single_error_ref)
        self.assertEqual(decode_barcode_8(self.single_error_8),
                         self.single_error_ref)

    def test_decode_barcode_8_two_error(self):
        # Should raise error when double error detected
        self.assertRaises(ValueError, decode_barcode_8,
                          self.double_error_1)
        self.assertRaises(ValueError, decode_barcode_8,
                          self.double_error_2)
        self.assertRaises(ValueError, decode_barcode_8,
                          self.double_error_3)
        self.assertRaises(ValueError, decode_barcode_8,
                          self.double_error_4)

if __name__ == '__main__':
    main()
Supplementary Data 2: demo output

Running demo...

---------------------------------------
Processing 25 sequences from 5 samples
---------------------------------------

--- processing demo sequence 0
read barcode : AACCATGC
-no error  barcode: AACCATGC
original sample id: Sample 1
sequence read  : TTTTTTTTTTTTTTTTTTTTTTTTT_0

--- processing demo sequence 1
read barcode : TCGTAGCA
-no error  barcode: TCGTAGCA
original sample id: Sample 2
sequence read  : TTTTTTTTTTTTTTTTTTTTTTTTT_1

--- processing demo sequence 2
read barcode : ACACCTCT
-no error  barcode: ACACCTCT
original sample id: Sample 3
sequence read  : TTTTTTTTTTTTTTTTTTTTTTTTT_2

--- processing demo sequence 3
read barcode : CTTCTAG
-no error  barcode: CTTCTAG
original sample id: Sample 4
sequence read  : TTTTTTTTTTTTTTTTTTTTTTTTT_3

--- processing demo sequence 4
read barcode : GGTAGCTT
-no error  barcode: GGTAGCTT
original sample id: Sample 5
sequence read  : TTTTTTTTTTTTTTTTTTTTTTTTT_4

--- processing demo sequence 5
read barcode : AACCATGC
*corrected barcode: AACCATGC
original sample id: Sample 1
sequence read  : AAAAAAAAAAAAAAAAAAAAAAAAAA_5

--- processing demo sequence 6
read barcode : TCGTAGCC
*corrected barcode: TCGTAGCA
original sample id: Sample 2
sequence read  : AAAAAAAAAAAAAAAAAAAAAAAAA_6

--- processing demo sequence 7
read barcode : ACACATCT
*corrected barcode: ACACCTCT
original sample id: Sample 3
sequence read  : AAAAAAAAAAAAAAAAAAAAAAAAA_7
--- processing demo sequence 8
read barcode  : TTTCCTAG
*corrected barcode: CTTCCTAG
original sample id: Sample 4
sequence read  : AAAAAAAAAAAAAAAAAAAAAA_8
--- processing demo sequence 9
read barcode  : GGTAGCTT
-no error  barcode: GGTAGCTT
original sample id: Sample 5
sequence read  : AAAAAAAAAAAAAAAAAAAAAAAA_9
--- processing demo sequence 10
read barcode  : AACCACGC
*corrected barcode: AACCATGC
original sample id: Sample 1
sequence read  : AAAAAAAAAAAAAAAAAAAAAA_10
--- processing demo sequence 11
read barcode  : TAGTAGCA
*corrected barcode: TCGTAGCA
original sample id: Sample 2
sequence read  : AAAAAAAAAAAAAAAAAAAAAAAA_11
--- processing demo sequence 12
read barcode  : ACACCTCT
-no error  barcode: ACACCTCT
original sample id: Sample 3
sequence read  : AAAAAAAAAAAAAAAAAAAAAA_12
--- processing demo sequence 13
read barcode  : CGTCCTAG
*corrected barcode: CTTCCTAG
original sample id: Sample 4
sequence read  : AAAAAAAAAAAAAAAAAAAAAA_13
--- processing demo sequence 14
read barcode  : GTTAGCTT
*corrected barcode: GGTAGCTT
original sample id: Sample 5
sequence read  : AAAAAAAAAAAAAAAAAAAAAA_14
--- processing demo sequence 15
read barcode  : AGCCATGC
*corrected barcode: AACCATGC
original sample id: Sample 1
sequence read  : AAAAAAAAAAAAAAAAAAAAAA_15
--- processing demo sequence 16
read barcode  : GCGTAGCA
*corrected barcode: TCGTAGCA
original sample id: Sample 2
sequence read  : AAAAAAAAAAAAAAAAAAAAAA_16
--- processing demo sequence 17
read barcode      : ACACCTCT
-no error barcode: ACACCTCT
original sample id: Sample 3
sequence read     : AAAAAAAAAAAAAAAAAAAAAAAA_17
    ---> processing demo sequence 18
read barcode      : ATTCCTAG
*corrected barcode: CTTCCTAG
original sample id: Sample 4
sequence read     : AAAAAAAAAAAAAAAAAAAAAAAA_18
    ---> processing demo sequence 19
read barcode      : GGTAGATT
*corrected barcode: GGTAGCTT
original sample id: Sample 5
sequence read     : AAAAAAAAAAAAAAAAAAAAAAAA_19
    ---> processing demo sequence 20
read barcode      : TACCATGC
! 2 bit error detected. skipping...
    ---> processing demo sequence 21
read barcode      : TCCTAGCA
! 2 bit error detected. skipping...
    ---> processing demo sequence 22
read barcode      : ACAGCTCT
! 2 bit error detected. skipping...
    ---> processing demo sequence 23
read barcode      : CTTCGTAG
! 2 bit error detected. skipping...
    ---> processing demo sequence 24
read barcode      : GCTAGCTT
! 2 bit error detected. skipping...
Demo done.