For more information on this example, see "The Markov chain Monte Carlo revolution" by Persi Diaconis, Bull. AMS 46 (2009). I'm using an English translation of Tolstoy’s "War and Peace" as the source of my word frequency data, and the US Constitution as my test example.

In [1]: using PyPlot

In [2]: C=include("Cipher.jl")

Out[2]: Main.Cipher

We compute a frequency matrix $M_{ij}$ of $i$-to-$j$ transitions for all letters of the alphabet $i$ and $j$, using a big source text. We then define the "pairwise energy" $\phi_{ij} = -\log M_{ij}$.

In [3]: src=C.readtext("/tmp/pg2600.txt");

In [4]: pot=C.PairPotential(src)

Out[4]: (::getfield(Main.Cipher, Symbol("#pot#6")){Float64,Array{Float64,2},Dict{Char,Int64}}) (generic function with 2 methods)

In [5]: pcolor(pot())
colorbar()
Out[5]: PyObject <matplotlib.colorbar.Colorbar object at 0x12b9723c8>

In [6]: pot('a','a')

Out[6]: -3.258096538021482

The "plaintext" is a copy of the US Constitution.

In [7]: ptext=C.readtext("/Users/kkylin/amalthea/constitution.txt");

In [8]: ptext[1:1000]

Out[8]: "provided by usconstitution.net-----------------------------note repealed text is not noted in this version. for an ... for electors of the most numerous branch of the state legislature. no person shall be a representative who shall n"

We now make a random substitution cipher and apply it to obtain a "ciphertext."

In [9]: rc = C.RandomCipher()

Out[9]: Dict{Char,Char} with 44 entries:
    'w' => '"'
    '7' => 'p'
    'o' => 'm'
    '5' => '9'
'h' => 't'
'i' => 's'
'r' => 'o'
'q' => '1'
';' => '!
'a' => ',',
'c' => 'e'
'p' => 'a'
'9' => 'g'
'x' => 'u'
'u' => 'y'
'd' => '3'
'e' => 'b'
'j' => '2'
's' => 'n'
'4' => '5'
',' => 'd'
'z' => 'r'
'0' => 'z'
'3' => 'v'
'n' => 'f'

In [10]: ctext = C.encrypt(rc,ptext);

In [11]: ctext[1:1000]

Out[11]: "aomcs3b3jh;jynemfn-s-y-smf4fb-wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwm-bjobab, b3j-bu-jsnjfm-j

We can recover the plaintext from the ciphertext.

In [12]: C.decrypt(rc,ctext)[1:1000]

Out[12]: "provided by usconstitution.net---------------------------------note repealed text is not

All that decrypt() does is invert the cipher (as a permutation) and apply it.

In [13]: irc=C.invertcipher(rc)

Out[13]: Dict{Char,Char} with 44 entries:
  'w' => '‘'
  '7' => '6'
  'o' => 'r'
  '5' => '4'
  'b' => '9'
  'i' => 'k'
  'r' => 'z'
  'q' => '2'
  ';' => 'y'
Based on this, for a string $s$ and a cipher $c$ we define the energy

$$E(c, s) = \sum_i \phi_{c(s_i), c(s_{i+1})}.$$ 

Note

$$e^{-E(c, s)} = \prod_i M_{c(s_i), c(s_{i+1})}.$$ 

So lower energy means the text is closer to having the correct statistics.

Here is the energy of the plaintext, computed two ways:

In [15]: e0=C.energy(pot,C.IdCipher(),ptext)
Out[15]: -423749.3414496625

In [16]: C.energy(pot,irc,ctext)
Out[16]: -423749.3414496625

Let’s run some Monte Carlo to try to decode this.

In [17]: @time cmin,emin,el=C.mh(pot,ctext,5000)

8.149655 seconds (313.50 k allocations: 27.242 MiB, 0.10% gc time)

Out[17]: (Dict('w' => '-') ', '7' => '2', 'o' => 'r', '5' => '8', 'h' => 'b', 'i' => 'k', 'r' => '\', 'q' => '9', ';' => 'y', 'a' => 'p', 'c' => 'v', 'p' => '7', '9' => '5', 'x' => '\', 'u' => 'x', 'd' => ',', 'e' => 'c', 'j' => ' ', 's' => 'i', '4' => '.', ',' => 'a', 'z' => '0', 'n' => 's', =>)

How good is the energy of the solution we found compared to the plaintext?
In [18]: emin/e0

Out[18]: 1.0003150226404838

In [19]: plot(el, label="MCMC sampler")
   plot([1,5000],[e0,e0], label="correct energy")
   plot([1,5000],[emin,emin], label="minimum found")
   legend()
   xlabel("step")
   ylabel("energy")

Out[19]:PyObject <matplotlib.text.Text object at 0x12eeb1fd0>

Does it really work? Let’s try decrypting the ciphertext with it.

In [20]: dctext = C.encrypt(cmin,ctext);

In [21]: dctext[1:1000]

Out[21]: "provided by usconstitution.net--------------------------note repealed text is not

Let’s count the percentage of characters that were correctly decoded.

In [22]: sum(map(==,dctext,ptext))/length(ptext)
Out[22]: 0.9958067952582788

The Monte Carlo is randomly initialized, so if we run it again we will get a different answer.

In [23]: @time cmin,emin,el=C.mh(pot,ctext,5000)

8.326295 seconds (58.72 k allocations: 14.062 MiB, 0.09% gc time)

Out[23]: (Dict('w'->'-','7'->'7','o'=>'t','s'->'0','h'=>' ','i'=>'\ ','r'=>'j','q'=>'!',';'=>'b'), -392983.3309713365, [-94284.0, -94284.0, -94284.0, -1.03959e5, -1.03959e5, -1.03959e5, -1.0487e5, ... -392637.0, -392637.0, -3.92457e5, -3.92457e5, -3.92457e5, -3.92457e5, -3.92457e5, -392463.0, -392463.0, -392463.0]

Hm, things look different this time: we don’t get as close to the plaintext energy.

In [24]: emin/e0

Out[24]: 0.9273957326445056

In [25]: plot(el,label="MCMC sampler")
   plot([[1,5000],[e0,e0],label="correct energy")
   plot([[1,5000],[emin,emin],label="minimum found")
   legend()
   xlabel("step")
   ylabel("energy")
Out[25]: PyObject <matplotlib.text.Text object at 0x12eeacef0>

    Does it work? Let’s try.

In [26]: dctext = C.encrypt(cmin,ctext);

In [27]: dctext[1:1000]

Out[27]: "ptrvodide.beusfrns o u ornkni -------------------------------nr ietipialide ix eosenr e