raster_tuning_HH

xyy

October 21, 2014

1 The HH model Used

It is the classical Hodgkin-Huxley (HH) neuron model. For neuron i, its membrane potential V_i obey

$$\begin{cases} C\dfrac{\mathrm{d}V_i}{\mathrm{d}t} = -(V_i - V_{\mathrm{Na}})G_{\mathrm{Na}}h_i m_i^3 - (V_i - V_{\mathrm{K}})G_{\mathrm{K}}n_i^4 - (V_i - V_{\mathrm{L}})G_{\mathrm{L}} + I_i^{\mathrm{input}} \\ \dfrac{\mathrm{d}m_i}{\mathrm{d}t} = (1-m_i)\alpha_m(V_i) - m_i\beta_m(V_i) \\ \dfrac{\mathrm{d}h_i}{\mathrm{d}t} = (1-h_i)\alpha_h(V_i) - h_i\beta_h(V_i) \\ \dfrac{\mathrm{d}n_i}{\mathrm{d}t} = (1-n_i)\alpha_n(V_i) - n_i\beta_n(V_i) \end{cases}$$

where

$$\begin{aligned} \alpha_n(V_i) &= \frac{0.1 - 0.01V_i}{\exp(1-0.1V_i) - 1} & \beta_n(V_i) &= 0.125\exp(-V_i/80) \\ \alpha_m(V_i) &= \frac{2.5 - 0.1V_i}{\exp(2.5-0.1V_i) - 1} & \beta_m(V_i) &= 4\exp\left(-V_i/18\right) \\ \alpha_h(V_i) &= 0.07\exp(-V_i/20) & \beta_h(V_i) &= \frac{1}{\exp(3-0.1V_i)+1} \end{aligned}$$

V_i, m_i, n_i, h_i, I_i^{input} are functions of t, and others are constants: $V_{\mathrm{Na}} = 115\,\mathrm{mV}$, $V_{\mathrm{K}} = -12\,\mathrm{mV}$, $V_{\mathrm{L}} = 10.6\,\mathrm{mV}$ (resting potential set to $0\,\mathrm{mV}$), $G_{\mathrm{Na}} = 120\,\mathrm{mS}\cdot\mathrm{cm}^{-2}$, $G_{\mathrm{K}} = 36\,\mathrm{mS}\cdot\mathrm{cm}^{-2}$, $G_{\mathrm{L}} = 0.3\,\mathrm{mS}\cdot\mathrm{cm}^{-2}$ and membrane capacity $C = 1\,\mu\mathrm{F}\cdot\mathrm{cm}^{-2}$.

The interaction between neurons and external inputs come from I_i^{input}

$$I_i^{\mathrm{input}} = I_i^{\mathrm{E}} + I_i^{\mathrm{I}}, \quad I_i^{\mathrm{E}} = -(V_i - V_G^{\mathrm{E}})G_i^{\mathrm{E}}, \quad I_i^{\mathrm{I}} = -(V_i - V_G^{\mathrm{I}})G_i^{\mathrm{I}}$$

I_i^{E}, I_i^{I} are excitatory and inhibitory input respectively, and V_G^{E}, V_G^{I} is their reversal potential. The conductances G_i^Q ($Q \in \{\mathrm{E}, \mathrm{I}\}$) evolves according to

$$\frac{\mathrm{d}G_i^Q}{\mathrm{d}t} = -\frac{G_i^Q}{\sigma_d^Q} + H_i^Q, \quad \frac{\mathrm{d}H_i^Q}{\mathrm{d}t} = -\frac{H_i^Q}{\sigma_r^Q} + \sum_k F_i^Q\delta(t - T_{i,k}^F) + \sum_{j\neq i} S_{ij}g(V_j^{\mathrm{pre}})$$

$$g(V_j^{\mathrm{pre}}) = 1/\left(1 + \exp(-(V_j^{\mathrm{pre}} - 85\,\mathrm{mV})/2)\right)$$

where F_i^Q is the strength of external input to neuron i, $T_{i,k}^F$ is its time of k-th input event, which is a Poisson process with rate μ_i. We call this term the Poisson input. S_{ij} is the coupling strength from j-th neuron to i-th neuron. V_j^{pre} is the (presynaptic) membrane potential of j-th neuron. σ_r^Q, σ_d^Q are the fast rising and slow decaying timescales in the α function. $V_G^{\mathrm{E}} = 65\,\mathrm{mV}$, $V_G^{\mathrm{I}} = -15\,\mathrm{mV}$, $\sigma_d^E = 0.5$, $\sigma_r^E = 3.0$, $\sigma_d^I = 0.5$, $\sigma_r^I = 7.0$.

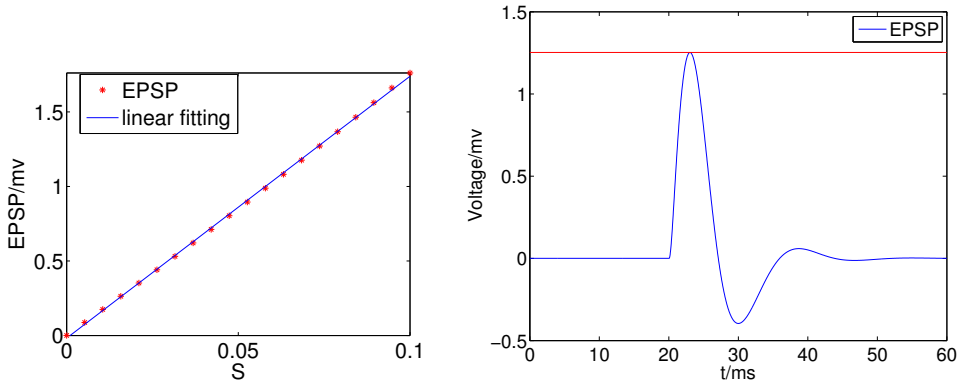
We use adjacency matrix $A = (A_{ij})$ to denote the neural network structure, i.e. $S_{ij} = A_{ij}S^{Q_iQ_j}$, and $S^{Q_iQ_j}$ is one of S^{EE}, S^{EI}, S^{IE}, S^{II}, depends on the type of corresponding neuron pair (E for excitatory, I for inhibitory). $A_{ij} \neq 0$ means there is a direct affection to i-th neuron from j-th neuron. When we talk about "homogeneous coupling", we mean A_{ij} equals either 1 or 0.

F, μ, A, $S^{Q_iQ_j}$, σ_r^Q, σ_d^Q are parameters relate to synaptic and input to neurons. For all neurons $F_i^{\mathrm{E}} = F$, $F_i^{\mathrm{I}} = 0$, $\mu_i = \mu$. During one simulation, these parameters are all constant.

The time delay due to long dendrite or axion are ignored.

In numerical simulation, we use explicit fourth-order Runge-Kutta method with time step 1/32 ms. The data samples (i.e. x_t, y_t) we used are voltages obtained in sampling rate 2 kHz. When we talk about spike train data, we mean $x_t = 1$ if $V_i(t)$ just pass through the threshold (10 mV in our case) from low to high, otherwise $x_t = 0$.

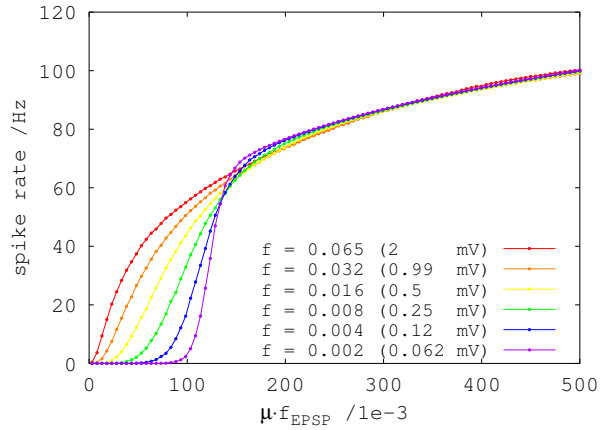
1.1 Few Basic Properties of the HH



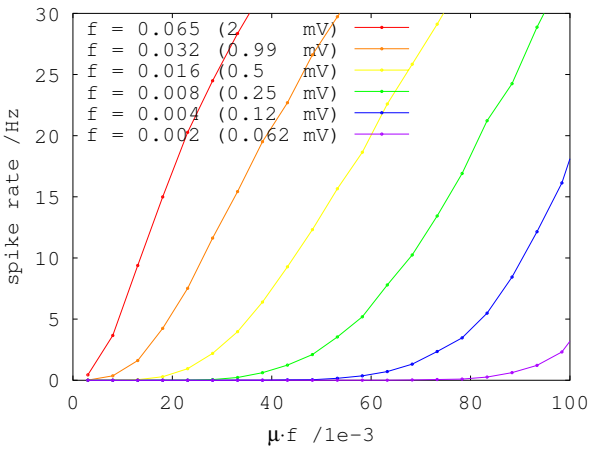
(a) (b)

Figure 1: (a) Excitatory postsynaptic potential (EPSP) - S^E relation. $V_{\mathrm{EPSP}} \approx 17\, S^{\mathrm{EE}}$ mV (b) typical EPSP voltage trace in this model.

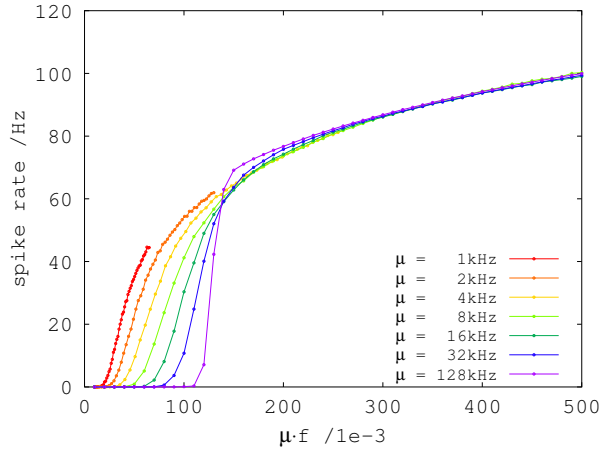
For inhibitory input and Poisson input, the curves are roughly the same (up to scalings) as Fig.(1). The relation between peak respond and model parameter is $V_{\mathrm{IPSP}} \approx -5.0\, S^{\mathrm{II}}$ mV, $V_{\mathrm{EPSP\text{-}Poisson}} \approx 31\, F^{\mathrm{E}}$ mV.



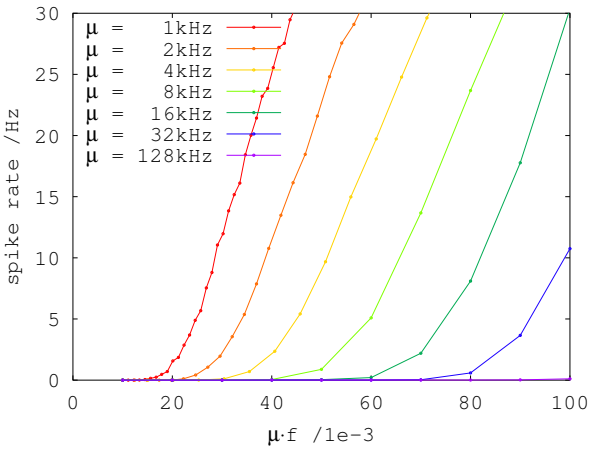
(a) Scan input rate μ.



(b) Scan input rate μ. Local



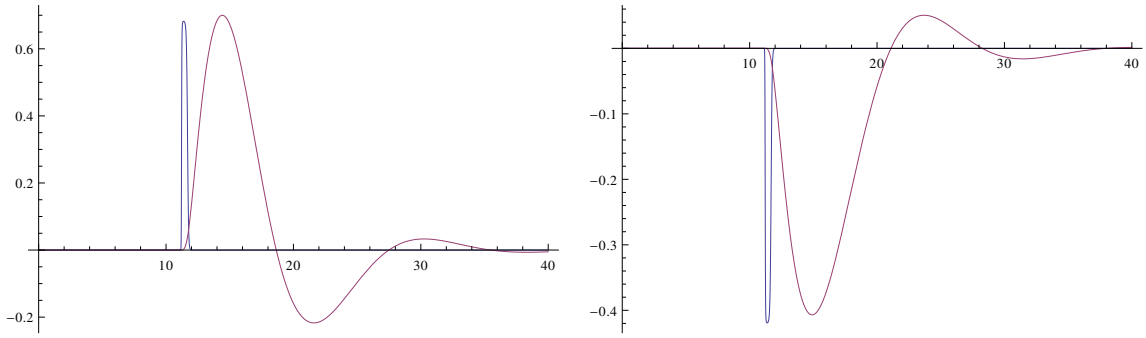
(c) Scan input strength f. Some curves terminate because EPSP exceed 2 mV.



(d) Scan input strength f. Local

Figure 2: Gain function under Poisson pulse drive.

For current input.
Slowest firing, ISI: 19.547 ms, rate: 51.159 Hz. (input current: 6.27)
Fast firing, ISI=8.544 ms, rate 117.03 Hz. (input current: 50)
Critical input current that firing will automatically disappeared: 6.264.



(a) EPSP ($s^{EE} = 0.05$)

(b) IPSP ($s^{II} = 0.1$)

Figure 3: The blue curve indicate the synaptic input.

Note that in this model, EPSP and IPSP will be affect by pre-synaptic neuron dynamics.

2 Simulator `raster_tuning_HH` changes

Compared current version (2014-08-10, changeset 45:836f718da9cf) to JYL version (2014-05-01 just after branch created, changeset 27:9efee6777a85)[1].

2.1 Speed improvement

Eliminate exponential calculation

Recall that there are 7 exponent calculation for each neuron at each time step

$$\begin{aligned}
\alpha_n(V_i) &= \frac{0.1 - 0.01V_i}{\exp(1 - 0.1V_i) - 1} & \beta_n(V_i) &= 0.125\exp(-V_i/80) \\
\alpha_m(V_i) &= \frac{2.5 - 0.1V_i}{\exp(2.5 - 0.1V_i) - 1} & \beta_m(V_i) &= 4\exp\left(-V_i/18\right) \\
\alpha_h(V_i) &= 0.07\exp(-V_i/20) & \beta_h(V_i) &= \frac{1}{\exp(3 - 0.1V_i) + 1}
\end{aligned}$$

$$g(V_j^{\text{pre}}) = 1/\left(1 + \exp(-(V_j^{\text{pre}} - 85\,\text{mV})/2)\right).$$

Reformulate the exponentials as following ($g(V_j^{\text{pre}})$ has no change)

$$e_1 = \exp(-0.1V_i), \quad e_2 = \sqrt{e_1}$$

$$\begin{aligned}
\exp(1 - 0.1V_i) &= e_1\exp(1) & \exp(-V_i/80) &= \sqrt{\sqrt{e_2}} \\
\exp(2.5 - 0.1V_i) &= e_1\exp(2.5) & \exp\left(-V_i/18\right) &= \exp\left(-V_i/18\right) \\
\exp(-V_i/20) &= e_2 & \exp(3 - 0.1V_i) &= e_1\exp(3)
\end{aligned}$$

So we only need to compute 3 exponentials, instead of 7 (the constant numbers like exp(1) will be calculated at compile time, instead of runtime).

Also note that there some expressions like $\frac{\exp(x)}{\exp(x)+a}$ in the origin code. They have been changed to $\frac{1}{1+a\exp(-x)}$, which is faster (if the compiler failed to optimize it), simpler and no precision lose.

The exponential elimination make the program 30% faster for 100 neuron case.

Eliminate unnecessary synaptic calculation

Neurons are not always firing. So don't do synaptic calculation at all if there is no spike. The original code contain this type of optimization, but only at halfway, like this

```
for neuron_passive = 1 to n
  for neuron_driving = 1 to n
    if (neuron_driving is firing)
      synaptic[neuron_passive] += v[neuron_driving]
                                  * matrix[neuron_passive][neuron_driving]
```

In whatever condition, there are $O(n^2)$ calculations there (also bad for CPU pipeline). Instead, write it this way

```
for neuron_driving = 1 to n
  if (neuron_driving is firing)
    for neuron_passive = 1 to n
      synaptic[neuron_passive] += v[neuron_driving]
                                  * matrix[neuron_passive][neuron_driving]
```

[1] https://bitbucket.org/bewantbe/ifsimu branch HH_jyl

Which cost much less if the network not spike too much.

With the above two eliminations, the program runs 1 times faster. (And synaptic calculation no more a hot spot for 100 neuron case)

Utilize SSE/AVX instructions

Thanks to the pipeline and Streaming SIMD (Single instruction, multiple data) Extensions (SSE) and Advanced Vector Extensions (AVX), modern CPU can calculate 2 or 4 (or even 8) multiplications (and potentially 2 or 4 or more additions at the same time) in ever clock cycle.

Use SSE/AVX instructions directly would need hand write Assembly code or equivalences (compiler intrinsics), or fine tune of compiler options, and probably in the end, no big success.

So use SSE instructions through third party package is a good compromise. I choose Eigen[2] here. Eigen performs automatic parallel also. Also note that, Eigen v3.2 is remarkably faster than Eigen v3.1.

Compile with -O3 (instead of -O2) and use a new compiler (GCC 4.9 here) can have few percentages of speed increment.

After analyze the hot spot of the program, and rewrite the hot spot in vector form (use Eigen), get 30%~40% more faster.

Use sparse representation for the adjacency matrix

Like before, now the synaptic loop become

```
for neuron_driving = 1 to n
  if (neuron_driving is firing)
    for edge = out_edge_list_of(neuron_driving)
      synaptic[edge.to_index()] += v[neuron_driving]
                                 * edge.value()
```

This make the 1000 neuron (sparse network) simulation 2~3 times faster. 100 neuron simulation 10% faster.

Further possible speed optimization

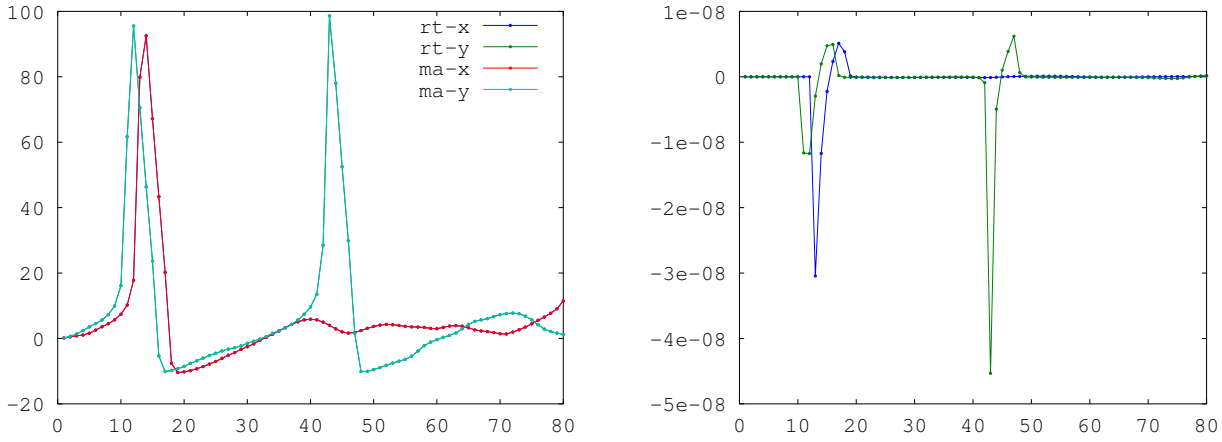
- Use highly optimized math function library for exp() calculation. (e.g. mkl, acml) The related code is still a hot spot.
- Review the hot spot and make it cache line friendly and pipeline friendly.

2.2 Program Correctness Verification

See matcode/prj_GC_clean/HH/test_HH.m for parameter details.

See matcode/prj_GC_clean/HH/HH_cal.nb for code to generate accurate results.

[2] http://eigen.tuxfamily.org/



(a) voltage trace

(b) voltage trace differences

Figure 4: Accuracy verification. x-axis is time in millisecond, y-axis is voltage in millivolt(-65 mV shifted). "rt" curves is generated by `raster_tuning_HH`, "ma" curve is generated by Mathematica using implicit Runge-Kutta with max time step 1/16 ms.

We can see that the error is below 10^{-9} (relatively) in short time.

3 Time Cost in Computation

Only for reference. They are computed on different computers.

Table 1: HHGC analysis time cost. There are ranges because there are different networks.

n	len/ms	Fr Rate(Hz)	HH simu	od_{max}	GC (sec)
50	10^6	32.0	0.838 h	40	74.2
50	10^6	12.2	0.509 h	40	81.6
100	10^6	33.7	2.200 h	40	117.0
100	10^6	12.5	1.203 h	40	120.3
200	10^6	36.0	11.97 h	40	1391~1608
200	10^6	13.9	5.51 h	40	1562
400	10^6	35.4~37.1	25.36 h~30.52 h	40	7256~8063
1000	10^6	?	3.4 d ~ 4.3 d	40	37.6 h