Fichiers mod du modèle

David Vincent

Août 2024

Table des matières

1 *kcc2.mod*

1.1 Script complet

```
TITLE K-Cl cotransporter KCC2

NEURON {
    SUFFIX kcc2
    USEION k READ ko, ki WRITE ik VALENCE 1
    USEION cl READ clo, cli WRITE icl VALENCE -1
    RANGE ik, icl, S, Vi, U
}

UNITS {
    (mV)    = (millivolt)
    (molar) = (1/liter)
    (mM)    = (millimolar)
    (um)    = (micron)
    (mA)    = (milliamp)
    (mol)   = (1)
    FARADAY = 96485.309 (faraday)
    PI      = (pi) (1)
}

PARAMETER {
    S  = 5654.87 (um2)
    Vi = 8913.48 (um3)
    U  = 0.0003  (mM/ms)
}

ASSIGNED {
    ik      (mA/cm2)
    icl     (mA/cm2)
    ko      (mM)
    ki      (mM)
    clo     (mM)
    cli     (mM)
}

BREAKPOINT {
    LOCAL rate
    rate = pumprate(ki, ko, cli, clo)
    ik   = rate
    icl  = -rate
}

FUNCTION pumprate(ki, ko, cli, clo) {
    pumprate = U*log((ki*cli)/(ko*clo))*(FARADAY*Vi/(S*1e4)
}
```

1.2 Informations générales

Ce fichier mod vient probablement du modèle suivant : *Changes of ionic concentrations during seizure transitions* [1]. Celui-ci est disponible sur ModelDB [2]. Le KCC2 est un cotransporteur cation-chlore (CCC) qui transporte, dans des conditions normales, un ion potassium et un ion chlorure hors de la cellule.

1.3 Bloc NEURON

```
NEURON {
    SUFFIX kcc2
    USEION k READ ko, ki WRITE ik VALENCE 1
    USEION cl READ clo, cli WRITE icl VALENCE -1
```

```
    RANGE ik, icl, S, Vi, U
}
```

Le bloc NEURON permet de déterminer si le mécanisme est un *'density mechanism'* ou un *'point process'*. Un *'density mechanism'* est uniformément distribué dans la section dans laquelle il est inséré. Un *'point process'* est un mécanisme qui est plutôt local et qui est inséré seulement dans un segment en particulier. C'est également dans ce bloc que les variables sont définies. Dans ce cas, la première ligne du bloc, 'SUFFIX kcc2' signifie que le mécanisme est un *'density mechanism'* dont le nom est 'kcc2'.

Les deux prochaines lignes sont de la forme :

$$\text{USEION x READ xo, xi WRITE ix VALENCE y}$$

où x est un ion, xo et xi sont les concentrations extracellulaire et intracellulaires de l'ion x, ix est un courant généré par l'ion x et y est la charge ionique de l'ion. 'READ xo, xi' signifie que les valeurs de ces variables sont obtenues et utilisées par le mécanisme. 'WRITE ix' signifie qu'un courant est généré par le mécanisme. 'VALENCE y' ne fait que définir dans le script la charge ionique de l'ion x. Dans le cas du mécanisme kcc2, les ions potassium et chlorure sont utilisés. Les concentrations des deux ions sont obtenus pour effectuer des calculs plus loin dans le script et des courants associés aux deux ions respectivement sont générés.

La dernière ligne du bloc permet de déclarer des variables 'RANGE'. Cela signifie que celles-ci seront accessibles au niveau du fichier python pour être enregistrées lors d'une simulation ou pour se faire assigner une valeur s'il s'agit d'un paramètre. 'ik' et 'icl' sont, comme mentionné précédemment, des courants de potassium et de chlore. 'S', 'Vi' et 'U' sont respectivement la surface de la section, le volume de la section et un coefficient de proportionnalité qui dicte la force du KCC2.

1.4 Bloc UNITS

```
UNITS {
    (mV)    = (millivolt)
    (molar) = (1/liter)
    (mM)    = (millimolar)
    (um)    = (micron)
    (mA)    = (milliamp)
    (mol)   = (1)
    FARADAY = 96485.309 (faraday)
    PI      = (pi) (1)
}
```

Ce bloc permet de créer des abréviations pour des unités qui seront utilisés plus loin dans le script. Le '(1)' correspond à aucune unité.

1.5 Bloc PARAMETER

```
PARAMETER {
    S  = 5654.87 (um2)
    Vi = 8913.48 (um3)
    U  = 0.0003  (mM/ms)
}
```

Les variables qui sont dans ce bloc sont des paramètres qui ne changent pas de valeur tout au long de la simulation. Pour assigner une valeur 'b' à un paramètre 'a' en supposant que le mécanisme est inséré dans la section 'Sec', la structure suivante doit être utilisée dans le fichier python : Sec.a = b. Dans le cas où aucune valeur n'est assignée à un paramètre dans le fichier python, la valeur de base (celle dans le fichier mod) sera utilisée. Entre parenthèse à côté de chaque paramètre sont les unités.

1.6 Bloc ASSIGNED

```
ASSIGNED {
    ik      (mA/cm2)
    icl     (mA/cm2)
```

```
    ko      (mM)
    ki      (mM)
    clo     (mM)
    cli     (mM)
}
```

Les variables 'ASSIGNED' sont des variables que le script utilise dans des calculs ou qui se font assigner une valeur calculée. Encore une fois, les unités des variables sont inscrites entre parenthèses à côté de chacune d'entre elles. Dans ce mécanisme, 'ik' et 'icl' se font assigner une valeur qui est calculée à partir des concentrations 'ko', 'ki', 'clo' et 'cli'.

1.7 Bloc BREAKPOINT

```
BREAKPOINT {
    LOCAL rate
    rate = pumprate(ki, ko, cli, clo)
    ik   = rate
    icl  = -rate
}
```

Le bloc 'BREAKPOINT' est le coeur du script. C'est dans ce bloc que sont effectués les calculs faits à chaque pas de temps. À la première ligne du bloc une variable locale au bloc et temporaire nommée 'rate' est créée grâce à la syntaxe 'LOCAL rate'. À cette variable est ensuite assignée une valeur calculée à partir de la fonction 'pumprate' qui prend en entrée les concentrations extracellulaires et intracellulaires des ions. Cette fonction est décrite dans la prochaine section. Les courants se font ensuite assignés une valeur absolue égale à 'rate', mais de signe opposé. Comme le potassium est un ion positif, un courant positif correspond à la sortie de potassium de la cellule (rate). Comme le chlorure est un ion négatif, pour que le courant corresponde à une sortie de chlore, le courant est négatif (-rate).

1.8 Bloc FUNCTION name(x1, x2, ...)

```
FUNCTION pumprate(ki, ko, cli, clo) {
    pumprate = U*log((ki*cli)/(ko*clo))*(FARADAY*Vi/(S*1e4)
}
```

Ce bloc permet de créer une fonction. Dans ce cas, la fonction permet de calculer la densité de courant créée par le KCC2. 'pumprate' est le nom de la fonction et entre parenthèse juste après ce nom se trouve les valeurs nécessaires au calcul de la fonction lorsque celle-ci est appelée.

2 *nkcc1.mod*

2.1 Script complet

```
TITLE N-K-Cl cotransporter NKCC1

NEURON {
    SUFFIX nkcc1
    USEION k READ ko, ki WRITE ik VALENCE 1
    USEION na READ nao, nai WRITE ina VALENCE 1
    USEION cl READ clo, cli WRITE icl VALENCE -1
    RANGE ina, ik, icl, S, Vi, U
}

UNITS {
    (mV)    = (millivolt)
    (molar) = (1/liter)
    (mM)    = (millimolar)
    (um)    = (micron)
```

```
    (mA)    = (milliamp)
    (mol)   = (1)
    FARADAY = 96485.309 (faraday)
    PI      = (pi) (1)
}

PARAMETER {
    S  = 5654.87 (um2)
    Vi = 8913.48 (um3)
    U  = 0.0001  (mM/ms)
}

ASSIGNED {
    ina     (mA/cm2)
    ik      (mA/cm2)
    icl     (mA/cm2)
    nao     (mM)
    nai     (mM)
    ko      (mM)
    ki      (mM)
    clo     (mM)
    cli     (mM)
}

BREAKPOINT {
    LOCAL rate
    rate = pumprate(nai, nao, ki, ko, cli, clo)
    ina  = rate
    ik   = rate
    icl  = -2*rate
}

FUNCTION pumprate(nai, nao, ki, ko, cli, clo) {
    pumprate = U*log((nai*ki*cli)/(nao*ko*clo))*(FARADAY*Vi/(S*1e4)
}
```

2.2 Informations générales

Le fichier a probablement été écrit par Justin Hamel en modifiant le fichier du KCC2. Le NKCC1 est un cotransporteur cation-chlore (CCC) qui transporte, dans des conditions normales, un ion potasium, un ion sodium et deux ions chlorure dans la cellule. La structure de ce fichier *mod* est la même que celle du mécanisme de KCC2 (section 1). Les seules différences sont que l'ion sodium est maintenant utilisé et que la fonction 'pumprate' considère maintenant les concentrations extra et intracellulaires de sodium. Également, le courant de chloreure correspond à deux fois la densité de courant créée par le mécanisme puisque le NKCC1 transporte deux ions chlorure dans la cellule.

3 *nakpump.mod*

3.1 Script complet

```
TITLE Na-K Pump

NEURON {
  SUFFIX nakpump
  USEION k READ ko WRITE ik VALENCE 1
  USEION na READ nai WRITE ina VALENCE 1
  RANGE ik, ina, km_k, km_na, imax, qna, qk
}

UNITS {
  (mV)  = (millivolt)
  (molar) = (1/liter)
```

```
  (mM)  = (millimolar)
  (um)  = (micron)
  (mA)  = (milliamp)
  (mol) = (1)
  FARADAY = 96485.309 (coul/mole)
  PI  = (pi) (1)
  R   = (k-mole)(joule/degC)
}

PARAMETER {
  km_k = 2  (mM)
  km_na = 10  (mM)
  imax = 0.013    (mA/cm2)
}

STATE { qna qk }

ASSIGNED {
  ik    (mA/cm2)
  ina   (mA/cm2)
  ko    (mM)
  nai   (mM)
  diam  (um)
  L   (um)
}

BREAKPOINT {
  SOLVE integrate METHOD sparse
}

INITIAL {
  qna=0
  qk=0
  ik = -2*imax*flux(nai,ko)
  ina = ik*-3/2
}

KINETIC integrate {
  ik = -2*imax*flux(nai,ko)
  ina = ik*-3/2

  COMPARTMENT diam*diam*PI/4 { qna qk }
  ~ qna << (-ina*PI*diam*(1e4)/FARADAY)
  ~ qk  <<  ( -ik*PI*diam*(1e4)/FARADAY)
}

FUNCTION flux(na,k) {
  flux = (1/((1+km_k/k)*(1+km_k/k)))*(1/((1+km_na/na)*(1+km_na/na)*(1+km_na/na))) : pas d'unite
}
```

3.2 Informations générales

Le code originel semble provenir du modèle suivant : *Changes of ionic concentrations during seizure transitions* [1]. Celui-ci est disponible sur ModelDB [2]. La pompe sodium potassium transporte deux potassiums dans la cellule et trois sodiums à l'extérieur de la cellule pour chaque ATP utilisé.

3.3 Bloc NEURON

```
NEURON {
  SUFFIX nakpump
  USEION k READ ko WRITE ik VALENCE 1
  USEION na READ nai WRITE ina VALENCE 1
```

```
  RANGE ik, ina, km_k, km_na, imax, qna, qk
}
```

La première ligne du bloc nous dit que le mécanisme est un *'density mechanism'* (voir section 1.3) nommé nakpump. Les deux prochaines lignes nous informent de l'utilisation des ions potassium et sodium. La concentration extracellulaire de potassium et la concentration intracellulaire de sodium sont obtenues. Un courant pour ces deux ions est généré par le mécanisme. La dernière ligne du bloc défini plusieurs variables 'RANGE' dont les courants mentionnés précédemment et d'autres paramètres/variables qui seront définis plus loin.

3.4 Bloc UNITS

```
UNITS {
  (mV)  = (millivolt)
  (molar) = (1/liter)
  (mM)  = (millimolar)
  (um)  = (micron)
  (mA)  = (milliamp)
  (mol) = (1)
  FARADAY = 96485.309 (coul/mole)
  PI  = (pi) (1)
  R   = (k-mole)(joule/degC)
}
```

Voir la section 1.4. Une nouveauté associé à ce bloc est la définition de la constante des gaz parfaits 'R'. Ici, '(k-mole)' ne correspond pas à kilomole, mais plutôt à la multiplication de la constante de Boltzman au nombre d'Avogadro [3]. Cela signifie que $\mathrm{R} = 8.313424\ \mathrm{J/(mol\ K)}$.

3.5 Bloc PARAMETER

```
PARAMETER {
  km_k = 2      (mM)
  km_na = 10      (mM)
  imax = 0.013    (mA/cm2)
}
```

Les trois paramètres initialisés ici sont nécessaires au calcul de la densité de courant créée par la pompe. 'imax' est la densité de courant maximale générée par la pompe.

3.6 Bloc STATE

```
STATE { qna qk }
```

Les variables qui sont dans le bloc state sont des variables d'une équation différentielle à résoudre. Ces variables vont être résolues dans un bloc 'KINETIC' ou 'DERIVATIVE'. Dans ce cas, qna et qk correspondent respectivement aux charges transportées hors de la cellule à cause du mouvement de sodium et aux charges transportées dans la cellule à cause du mouvement de potassium.

3.7 Bloc ASSIGNED

```
ASSIGNED {
  ik    (mA/cm2)
  ina   (mA/cm2)
  ko    (mM)
  nai   (mM)
  diam  (um)
  L   (um)
}
```

Ce sont les variables utilisées dans des calculs ou qui se font assigner une valeur par le script. 'diam' et 'L' correspondent au diamètre et à la longueur de la section. Ces variables n'ont pas besoin d'être défini dans le bloc 'NEURON' puisque leur nom est toujours réservé aux dimensions du neurone.

3.8 Bloc BREAKPOINT

```
BREAKPOINT {
  SOLVE integrate METHOD sparse
}
```

À chaque pas de temps, 'BREAKPOINT' dit à NEURON de résoudre le bloc 'KINETIC integrate' en utilisant la méthode 'sparse'. Sans que je ne puisse dire pour quelle raison 'sparse' est utilisé, c'est souvent cette méthode qui est choisie pour les blocs 'KINETIC'.

3.9 Bloc INITIAL

```
INITIAL {
  qna = 0
  qk  = 0
  ik  = -2*imax*flux(nai,ko)
  ina = ik*-3/2
}
```

Ce bloc permet de donner une valeur initiale aux variables 'ASSIGNED' et 'STATE' et d'effectuer des calculs avant le début de la simulation. Dans ce cas, les charges transportées par le mécanisme sont initialement mis à 0 et les courants causés par la pompe dans les concentrations ioniques initiales sont calculés. Pour ce faire, la fonction 'flux(x1,x2)' est appelée. Cette fonction est détaillée plus loin.

3.10 Bloc KINETIC integrate

```
KINETIC integrate {
  ik = -2*imax*flux(nai,ko)
  ina = ik*-3/2

  COMPARTMENT diam*diam*PI/4 { qna qk }
  ~ qna << (-ina*PI*diam*(1e4)/FARADAY)
  ~ qk  << ( -ik*PI*diam*(1e4)/FARADAY)
}
```

Dans ce bloc, il se trouve les équations résolues par le bloc 'BREAKPOINT'. Les courants de potassium et de sodium sont d'abord calculés. Ces calculs n'aurait en soi pas besoin d'être dans ce bloc. Toutefois, ils sont ensuite utilisés dans des équations qui vont être résolues. La ligne débutant par 'COMPARTMENT' défini l'aire de la région transeversale dans laquelle les charges sortent/entrent pour chaque segment. Les deux prochaines lignes débutent par '∼'. Cela indique que ce qui est écrit représente un schéma cinétique. Le '«' représente un flux explicite. Donc, dans ce cas, l'équation représente un flux de charge vers l'extérieur ou l'intérieur du compartiment.

4 *leak.mod*

4.1 Script complet

```
TITLE leak current :passive electrical properties

NEURON {
  SUFFIX leak
```

```
  USEION k READ ek WRITE ik VALENCE 1
  USEION na READ ena WRITE ina VALENCE 1
  USEION cl READ ecl WRITE icl VALENCE -1
    NONSPECIFIC_CURRENT ifix
  RANGE gk, ik, gna, ina, gcl, icl, ecl, gfix, ifix, gnaother
    RANGE qk, qna, qcl
}

UNITS {
  (mV) = (millivolt)
    (mA) = (milliamp)
    PI    = (pi) (1)
  FARADAY   = 96485.309 (coul/mole)
}

PARAMETER {
  gk   = 5e-5 (mho/cm2)  :potassium leak conductance
  gna  = 1e-5 (mho/cm2)  :sodium leak conductance : 1
  gnaother = 1e-5 (mho/cm2) :sodium other conductance
  gcl  = 0.05e-5 (mho/cm2)  :chloride leak conductance : 1
  gfix = 0 (mho/cm2)  :fixed leak conductance
}

ASSIGNED {
  v (mV)
    v_init (mV)
  ik (mA/cm2)
  ek (mV)
  ina (mA/cm2)
  ena (mV)
  icl (mA/cm2)
  ecl (mV)
  ifix (mA/cm2) :fixed leak current
    diam (um)
}

BREAKPOINT {
  ik = gk*(v-ek)
  ina = gna*(v-ena) + gnaother*(v-ena)
  icl = gcl*(v-ecl)
  :ifix = gfix*(v-v_init)
  ifix = gfix*(v+70)
    SOLVE integrate METHOD sparse
}

STATE { qk qna qcl}

INITIAL {
  ik = gk*(v-ek)
  ina = gna*(v-ena) + gnaother*(v-ena)
  icl = gcl*(v-ecl)
  qk = 0
  qna = 0
  qcl = 0
}

KINETIC integrate {
  COMPARTMENT diam*diam*PI/4 { qna qk qcl }
  ~ qna << ((-ina*diam)*PI*(1e4)/FARADAY )
  ~ qk  << ((-ik*diam)*PI*(1e4)/FARADAY )
  ~ qcl << ((-icl*diam)*PI*(1e4)/FARADAY )
}
```

4.2 Informations générales

Le code originel semble provenir du modèle suivant : *Changes of ionic concentrations during seizure transitions* [1]. Celui-ci est disponible sur ModelDB [2]. Les canaux de fuites sont des canaux passifs qui laissent en tout temps passer l'ion en question.

4.3 Bloc NEURON

```
NEURON {
    SUFFIX leak
    USEION k READ ek WRITE ik VALENCE 1
    USEION na READ ena WRITE ina VALENCE 1
    USEION cl READ ecl WRITE icl VALENCE -1
    NONSPECIFIC_CURRENT ifix
    RANGE gk, ik, gna, ina, gcl, icl, ecl, gfix, ifix, gnaother
    RANGE qk, qna, qcl
}
```

La première ligne indique que le mécanisme est un *'density mechanism'* nommé leak. Les trois prochaines lignes nous informent de l'utilisation des ions potassium, sodium et chlorure. Les potentiels réversibles des ions sont obtenus et des courants sont générés pour chacun des ions. La cinquième ligne indique que le mécanisme génère un courant qui n'est pas spécifique à aucun ion. La dernière ligne du bloc défini plusieurs variables 'RANGE' dont les courants mentionnés précédemment et d'autres paramètres/-variables qui seront définis plus loin.

4.4 Bloc UNITS

```
UNITS {
    (mV) = (millivolt)
    (mA) = (milliamp)
    PI    = (pi) (1)
    FARADAY   = 96485.309 (coul/mole)
}
```

Voir la section 1.4.

4.5 Bloc PARAMETER

```
PARAMETER {
  gk   = 5e-5 (mho/cm2)  :potassium leak conductance
  gna  = 1e-5 (mho/cm2)  :sodium leak conductance : 1
  gnaother = 1e-5 (mho/cm2) :sodium other conductance
  gcl  = 0.05e-5 (mho/cm2)  :chloride leak conductance : 1
  gfix = 0 (mho/cm2)  :fixed leak conductance
}
```

Les conductances associées à chaque canal de fuite sont initialisées ici. Ces valeurs sont habituellement déterminées au niveau du fichier python.

4.6 Bloc ASSIGNED

```
ASSIGNED {
    v (mV)
    v_init (mV)
    ik (mA/cm2)
    ek (mV)
    ina (mA/cm2)
    ena (mV)
    icl (mA/cm2)
    ecl (mV)
```

```
    ifix (mA/cm2) :fixed leak current
    diam (um)
}
```

Voir 3.7. Le potentiel de membrane ('v'), les potentiels réversibles des ions ('ek', 'ena' et 'ecl') et le diamètre de la section ('diam') sont des variables utilisées pour des calcul dans le bloc 'BREAKPOINT'. Les courants se font tous assignés une valeur calculée ('ik', 'ina', 'icl' et 'ifix').

4.7 Bloc BREAKPOINT

```
BREAKPOINT {
    ik = gk*(v-ek)
    ina = gna*(v-ena) + gnaother*(v-ena)
    icl = gcl*(v-ecl)
    :ifix = gfix*(v-v_init)
    ifix = gfix*(v+70)
    SOLVE integrate METHOD sparse
}
```

À chaque pas de temps, les courants sont calculées ici et le bloc 'KINETIC integrate' est résolu par NEURON en utilisant la méthode d'intégration 'sparse'.

4.8 Bloc STATE

```
STATE { qk qna qcl}
```

Ces variables correspondent aux charges associées aux différents ions qui sortent de chaque compartiment. Ces variables sont résolues dans le bloc BREAPOINT et suivent les équations du bloc 'KINETIC integrate'.

4.9 Bloc INITIAL

```
INITIAL {
  ik = gk*(v-ek)
  ina = gna*(v-ena) + gnaother*(v-ena)
  icl = gcl*(v-ecl)
  qk = 0
  qna = 0
  qcl = 0
}
```

Le bloc contient les valeurs initiales données aux variables 'ASSIGNED' et 'STATE'. Les charges sont initialisées à 0 et les courants sont calculés pour les conditions initiales de la simulation.

4.10 Bloc KINETIC integrate

```
KINETIC integrate {
  COMPARTMENT diam*diam*PI/4 { qna qk qcl }
  ~ qna << ((-ina*diam)*PI*(1e4)/FARADAY )
  ~ qk  << ((-ik*diam)*PI*(1e4)/FARADAY )
  ~ qcl << ((-icl*diam)*PI*(1e4)/FARADAY )
}
```

Il s'agit de la même chose que la section 3.10. Toutefois, les charges de chlorure sont aussi considérées cette foi-ci.

5 *puff.mod*

5.1 Script complet

```
TITLE GABA neurotransmitters injection

NEURON {
  POINT_PROCESS gabpuff
  USEION gab READ gabo VALENCE 0
  USEION mess READ messi WRITE messi VALENCE 0
    RANGE gabo, time0, GABAINIT, GABAdur, temp
}

UNITS {
  (mM)    = (milli/liter)
}

PARAMETER {
  GABAdur = 0.1   (ms)
}

ASSIGNED {
  gabo    (mM)    : transmitter concentration
  messi   (mM)
  time0   (ms)
  GABAINIT  (mM)
  temp
}

INITIAL {
  gabo   = 0    (mM)
  GABAINIT = 0  (mM)
  messi = 0     (mM)
  temp = 0
}

BREAKPOINT {
  if (GABAINIT > 0) {
    if (temp == 0) {
      messi = GABAINIT
      temp = 1
    }
  }
  else { messi = 0 }
}


NET_RECEIVE(weight(mM), nspike) {
  : an onset event (generated by NetStim) always has an implicit argument called flag which is set
     to 0
  if (flag == 0) {
    nspike = nspike + 1
    time0 = t
    GABAINIT  = weight
    : come again in Cdur with flag = current value of nspike (selfevent generated with delay Cdur
    & flag=nspike)
    net_send(GABAdur, nspike)
  }
  if (flag == nspike) {
    : if this associated with last spike then turn off
    GABAINIT = 0
  }
}
```

5.2 Information générale

Une partie de ce code provient du modèle suivant : *Effects of Chloride accumulation and diffusion on GABAergic transmission* [4]. Celui-ci est disponible sur ModelDB [5]. Ce mécanisme modélise l'éjection de GABA à un endroit sur la dendrite. Cette éjection est simplifiée. Elle représente une concentration de GABA extracellulaire écrite à un certain moment et un certain endroit de la simulation.

5.3 Bloc NEURON

```
NEURON {
  POINT_PROCESS gabpuff
  USEION gab READ gabo VALENCE 0
  USEION mess READ messi WRITE messi VALENCE 0
    RANGE gabo, time0, GABAINIT, GABAdur, temp
}
```

La première ligne du bloc indique que le mécanisme est un *'point process'* nommé gabpuff. Les deux prochaines lignes indiquent l'utilisation des "ions" 'gab' et 'mess'. Ce ne sont en réalité pas des ions, mais pour que NEURON puisse gérer la concentration de GABA à chaque segment, la fonctionnalité de 'USEION' est utilisée. 'gab' est associé au GABA. 'mess' est aussi associé au GABA, mais il ne s'agit que d'un messager. Ce messager permet de faire le lien entre l'éjection de GABA et le mécanisme qui gère les concentrations : *iondif.mod* (prochaine section).

La dernière ligne du bloc défini plusieurs variables 'RANGE' dont la concentration extracellulaire de GABA ('gabo'), la concentration initiale éjecté ('GABAINIT'), la durée de la GABA puff ('GABAdur'), le temps auquel la GABA puff survient ('time0') et une variable booléenne (valeur de 0 ou 1, 'temp'). Cette dernière variable permet d'écrire une seule fois la concentration éjectée par la puff ('messi'). Une fois que cela est fait, la concentration du messager est maintenu à 0.

5.4 Bloc PARAMETER

```
PARAMETER {
  GABAdur = 0.1   (ms)
}
```

Ce paramètre correspond à la durée de la GABA puff.

5.5 Bloc ASSIGNED

```
ASSIGNED {
  gabo    (mM)    : transmitter concentration
  messi   (mM)
  time0   (ms)
  GABAINIT  (mM)
  temp
}
```

Voir la section 3.7. Toutes les variables du bloc se font assigner des valeurs.

5.6 Bloc INITIAL

```
INITIAL {
  gabo   = 0    (mM)
  GABAINIT = 0  (mM)
  messi = 0     (mM)
  temp = 0
}
```

Initialise toutes les concentrations initiales à 0 et la variable 'temp' à 0.

5.7 Bloc BREAKPOINT

```
BREAKPOINT {
  if (GABAINIT > 0) {
    if (temp == 0) {
      messi = GABAINIT
      temp = 1
    }
  }
  else { messi = 0 }
}
```

Les conditions de ce bloc font en sorte que rien n'est effectué tant que la GABA puff n'a pas eu lieu. Lorsque la simulation arrive au moment de la GABA puff, GABAINIT se fait assigner une valeur (prochaine section) et devient plus grand que 0. La condition de la première ligne du bloc est alors respectée. S'il s'agit de la première fois que la condition est respectée, la deuxième condition à la deuxième ligne sera elle aussi respectée et la concentration de GABAINIT sera assignée au messager ('messi'). Une valeur de 1 sera ensuite donné à 'temp'. Ce faisant, au prochain pas de temps, la condition de la première ligne sera encore respectée, mais celle de la seconde ne le sera plus et une valeur de 0 sera attribuée à 'messi'.

Tout cela fait en sorte que 'messi' sera non nulle à un seul instant de la simulation.

5.8 Bloc NET_RECEIVE

```
NET_RECEIVE(weight(mM), nspike) {
  : an onset event (generated by NetStim) always has an implicit argument called flag which is set
     to 0
  if (flag == 0) {
    nspike = nspike + 1
    time0 = t
    GABAINIT  = weight
    : come again in Cdur with flag = current value of nspike (selfevent generated with delay Cdur
    & flag=nspike)
    net_send(GABAdur, nspike)
  }
  if (flag == nspike) {
    : if this associated with last spike then turn off
    GABAINIT = 0
  }
}
```

Ce bloc fait le lien entre le fichier python où les caractéristiques de la GABA puff sont décidées et le mécanisme du fichier mod. Lorsque l'évènement est généré par la fonction NetStim par python, une variable implicite nommée 'flag' est initialisée à 0. La première condition est ainsi respectée et les variables 'time0' et 'GABAINIT' reçoivent leur valeur.

6 *iondif.mod*

6.1 Script complet

```
COMMENT

Chloride accumulation and diffusion with decay (time constant tau) to resting level cli0.
The decay approximates a reversible chloride pump with first order kinetics.
To eliminate the chloride pump, just use this hoc statement
To make the time constant effectively "infinite".
tau and the resting level are both RANGE variables

Diffusion model is modified from Ca diffusion model in Hines & Carnevale:
Expanding NEURON with NMODL, Neural Computation 12: 839-851, 2000 (Example 8)

```

```
Nannuli correspond to the number of shells*

ENDCOMMENT

NEURON {
  SUFFIX iondifus
  USEION cl READ icl WRITE cli, clo VALENCE -1
  USEION hco3 READ hco3i, hco3o VALENCE -1
  USEION na READ ina WRITE nai, nao VALENCE 1
  USEION k READ ik WRITE ki, ko VALENCE 1
  USEION gab READ gabo WRITE gabo VALENCE 0
  USEION mess READ messi VALENCE 0
  GLOBAL vrat, DGab, taugaba, fhspace, areaext, DCl   :vrat must be GLOBAL
  GLOBAL tau, clipip, kipip, naipip
  RANGE cli0, clo0, nai0, nao0, ki0, ko0, egaba, delta_egaba, init_egaba, ehco3_help, ecl_help
  RANGE gabo0, temp
  RANGE clamp, hco3i0, hco3o0
}

DEFINE Nannuli 4

UNITS {
  (molar) = (1/liter)
  (mM) = (millimolar)
  (um) = (micron)
  (mA) = (milliamp)
  (mV)    = (millivolt)
  FARADAY = (faraday) (10000 coulomb)
  PI = (pi) (1) : le 1 entre parenthese est necessaire pour dire que c est adimensionnel
  F = (faraday) (coulombs)
  R = (k-mole)  (joule/degC)
}

PARAMETER {
  bath  = 0   (mM)
  DCl   = 2   (um2/ms) : Kuner & Augustine, Neuron 27: 447
  DK    = 1.96  (um2/ms)
  DNa   = 1.3   (um2/ms) :0.1 in original
  DGab  = 0.6 (um2/ms)
  fhspace = 0.03  (um)
  tau   = 100   (ms)
  cli0  = 3.5   (mM) : 8 mM in original file
  clo0  = 133.5 (mM)
  nai0  = 10  (mM)
  gabo0   = 0   (mM)
  nao0  = 147.5 (mM)
  ki0   = 135   (mM)
  ko0   = 3.5   (mM)
  hco3i0  = 16  (mM)
  hco3o0  = 26  (mM)
  P_help  = 0.18
  celsius = 37    (degC)
  taugaba = 100 (ms)
  clamp   = 0
  clipip  = 8   (mM)
  kipip   = 140 (mM)
  naipip  = 12  (mM)
}

ASSIGNED {
  diam  (um)
  icl   (mA/cm2)
  cli   (mM)
  clo   (mM)
  ina   (mA/cm2)
```

```
  nai   (mM)
  nao   (mM)
  ik  (mA/cm2)
  ki  (mM)
  ko    (mM)
  hco3i (mM)
  hco3o (mM)
  vrat[Nannuli] : numeric value of vrat[i] equals the volume
      : of annulus i of a 1um diameter cylinder
      : multiply by diam^2 to get volume per um length
  areaext
  areapip
  egaba   (mV)
  ehco3_help  (mV)
  ecl_help  (mV)
  init_egaba  (mV)
  delta_egaba (mV)
  messi     (mM)
  temp
}

STATE {
  : cl[0] is equivalent to cli
  : cl[] are very small, so specify absolute tolerance
  cl[Nannuli] (mM) <1e-10>
  na[Nannuli] (mM) <1e-10>
  k[Nannuli]  (mM) <1e-10>
  gabo    (mM) <1e-10>
}

BREAKPOINT {
    if (messi > 0) {
      if (temp == 0) {
        gabo = messi
        temp = 1
      }
    }

    SOLVE state METHOD sparse
    ecl_help = log(cli/clo0)*(1000)*(celsius + 273.15)*R/F
    egaba = P_help*ehco3_help + (1-P_help)*ecl_help
    delta_egaba = egaba - init_egaba
}

LOCAL factors_done

INITIAL {
  if (factors_done == 0) {    : flag becomes 1 in the first segment
    factors_done = 1  : all subsequent segments will have
    factors()   : vrat = 0 unless vrat is GLOBAL
  }

  temp = 0
  messi = 0
  cli = cli0
  clo = clo0
  nai = nai0
  nao = nao0
  ki = ki0
  ko = ko0
  gabo = 0
  hco3i = hco3i0
  hco3o = hco3o0
  FROM i=0 TO Nannuli-1 {
    cl[i] = cli
```

```
    na[i] = nai
    k[i] = ki
  }

  ehco3_help = log(hco3i/hco3o)*(1000)*(celsius + 273.15)*R/F
  ecl_help = log(cli/clo0)*(1000)*(celsius + 273.15)*R/F
  egaba = P_help*ehco3_help + (1-P_help)*ecl_help
  init_egaba = egaba
  delta_egaba = egaba - init_egaba
}

LOCAL frat[Nannuli] : scales the rate constants for model geometry

PROCEDURE factors() {
  LOCAL r, dr2, rgab
  r = 1/2     : starts at edge (half diam), diam = 1, length = 1
  dr2 = r/(Nannuli-1)/2 : full thickness of outermost annulus,
        : half thickness of all other annuli
  vrat[0] = 0
  frat[0] = 2*r   : = diam

  FROM i=0 TO Nannuli-2 {
    vrat[i] = vrat[i] + PI*(r-dr2/2)*2*dr2  : interior half
    r = r - dr2
    frat[i+1] = 2*PI*r/(2*dr2)  : outer radius of annulus Ai+1/delta_r=2PI*r*1/delta_r
            : div by distance between centers
    r = r - dr2
    vrat[i+1] = PI*(r+dr2/2)*2*dr2  : outer half of annulus
  }

  rgab = (diam+fhspace)/2
  areaext = PI*rgab*rgab - PI*(diam*diam)/4
  areapip = PI*2*2
}

KINETIC state {
  COMPARTMENT areaext {gabo}
  LONGITUDINAL_DIFFUSION DGab*areaext  {gabo}
  ~ gabo <-> bath (1/(taugaba), 1/(taugaba))

  COMPARTMENT i, diam*diam*vrat[i] {cl na k}
  LONGITUDINAL_DIFFUSION i, DCl*diam*diam*vrat[i] {cl}
  LONGITUDINAL_DIFFUSION i, DNa*diam*diam*vrat[i] {na}
  LONGITUDINAL_DIFFUSION i, DK*diam*diam*vrat[i] {k}

  ~ cl[0] << ((icl*PI*diam/FARADAY))
  ~ na[0] << ((-ina*PI*diam/FARADAY))
  ~ k[0] << ((-ik*PI*diam/FARADAY))

  FROM i=0 TO Nannuli-2 {
    ~ cl[i] <-> cl[i+1] (DCl*frat[i+1], DCl*frat[i+1])
    ~ na[i] <-> na[i+1] (DNa*frat[i+1], DNa*frat[i+1])
    ~ k[i] <-> k[i+1] (DK*frat[i+1], DK*frat[i+1])
  }

  if (clamp == 0) {
    cli = cl[0]
    nai = na[0]
    ki  = k[0]
  }
  else {
    ~ cl[0] <-> clipip (1/(tau), 1/(tau))
    ~ na[0] <-> naipip (1/(tau), 1/(tau))
    ~ k[0] <-> kipip (1/(tau), 1/(tau))
    cli = cl[0]
```

```
    nai = na[0]
    ki  = k[0]
  }

  clo = clo0
  nao = nao0
  ko = ko0
}
```

6.2 Informations générales

Le code originel provient de l'exemple 8 dans [3]. Pour une description complète du script, voir [3]. Le texte qui suit portera sur les modifications apportées au fichier original. Ce mécanisme gère les concentrations d'ions dans la cellule, la diffusion radiale des ions dans le neurone ainsi que la diffusion longitudinale d'un compartiment à l'autre. Les ajouts faits au code de base sont énumérés dans les sections suivantes.

6.3 Diffusion du GABA extracellulaire

6.3.1 Bloc NEURON

```
NEURON {
    ...
    USEION gab READ gabo WRITE gabo VALENCE 0
    USEION mess READ messi VALENCE 0
    ...
    GLOBAL vrat, DGab, taugaba, fhspace, areaext, DCl   :vrat must be GLOBAL
    ...
    RANGE gabo0, temp
    RANGE clamp, hco3i0, hco3o0
 }
```

Les premières lignes indiquent l'utilisation des "ions" 'gab' et 'mess' (voir section 5.3). Les deux prochaines lignes définissent les paramètres associés à la diffusion du GABA extracellulaire. 'DGab' correspond au coefficient de diffusion du GABA, 'taugaba' correspond au taux d'échange avec le bain (région où la concentration de GABA est 0), 'fhspace' correspond à une distance radiale (à partir de la surface du neurone) qui permet de définir la région transversale dans laquelle le GABA diffuse et 'areaext' correspond à cette région transversale.

Les deux dernières lignes du bloc permettent la définition de cinq variables 'RANGE'. La variable 'gabo0' permet de définir la concentration initiale de GABA extracellulaire. Cette variable n'est actuellement pas utilisée puisque la concentration initiale est toujours nulle dans le cadre des simulation. Comme pour le mécanisme précédent, la variable 'temp' est une variable booléenne ayant une valeur de 0 ou 1. Son utilité sera mis de l'avant plus loin. 'clamp' est une autre variable ayant comme valeur 0 ou 1 et permet d'indiquer la présence d'une *voltage clamp* sur le soma (prochaine section). Finalement, les deux dernières variables, 'hco3i0' et 'hco3o0', correspondent respectivement à aux concentrations intracellulaire et extracellulaire de HCO3-. Ces concentrations sont considérées comme fixes dans le modèle.

6.3.2 Bloc PARAMETER

```
PARAMETER {
    bath  = 0   (mM)
    ...
    DGab  = 0.6 (um2/ms)
    fhspace = 0.03  (um)
    tau   = 100   (ms)
    ...
    gabo0   = 0   (mM)
    ...
    hco3i0  = 16  (mM)
    hco3o0  = 26  (mM)
    ...
    taugaba = 100 (ms)
```

```
    ...
}
```

Initialisation des paramètres dont la majorité ont été énumérés dans la section 6.3.1. Les valeurs données aux paramètres sont sans importance ici pusique celles-ci sont décidées au niveau du script python mis à part pour le paramètre 'bath'. Ce dernier a une valeur de 0 et indique qu'à partir d'une certaine région autour du neurone, la concentration de GABA est de 0.

6.3.3 Bloc ASSIGNED

```
ASSIGNED {
    ...
    hco3i (mM)
    hco3o (mM)
    ...
    areaext
    ...
    messi     (mM)
    temp
}
```

Les variables 'ASSIGNED' sont identifiées ici. 'hco3i' et 'hco3o' sont utilisées plus loin dans des calculs. 'areaext' se fait assignée une valeur plus loin et la valeur de 'messi', elle, est assignée à une autre variable plus loin. La valeur de 'temp' passe de 0 à 1 lors de l'éxécution du script.

6.3.4 Bloc STATE

```
STATE {
    ...
    gabo    (mM) <1e-10>
}
```

'gabo' qui correspond à la concentration extracellulaire de GABA est défini comme étant une nouvelle variable 'STATE'. Cela permet de résoudre 'gabo' à chaque segment de la cellule et, ainsi, de considérer sa diffusion.

6.3.5 Bloc BREAKPOINT

```
BREAKPOINT {
    if (messi > 0) {
      if (temp == 0) {
        gabo = messi
        temp = 1
      }
    }
    ...
}
```

La première ligne indique une première condition nécessaire qui est que la concentration du messager ('messi'), contrôlée par le mécanisme de *puff.py*, soit plus grande que 0. Si c'est le cas et que c'est la première fois que la condition est respectée, la seconde condition est aussi respectée (temp = 0). Alors, la concentration extracellulaire de GABA, 'gabo', se fait assignée la valeur du messager. La variable 'temp' se fait également assignée une valeur de 1. Cela permet d'assurer que 'gabo' ne se fait assignée la valeur du messager qu'une seule et unique fois.

6.3.6 Bloc INITIAL

```
INITIAL {
    ...
    temp = 0
    messi = 0
    ...
    gabo = 0
    hco3i = hco3i0
    hco3o = hco3o0
    ...
}
```

Les concentrations associées au GABA sont initialisées à 0. Les concentrations extracellulaire et intracellulaire de HCO3- sont initialisées à leur valeur initiale déterminée dans le script python. La variable 'temp' est initialisée à 0.

6.3.7 Bloc PROCEDURE factor()

```
PROCEDURE factors() {
    LOCAL r, dr2, rgab
    ...
    rgab = (diam+fhspace)/2
    areaext = PI*rgab*rgab - PI*(diam*diam)/4
    ...
}
```

Cette procédure est exécutée une seule fois à l'exécution du script et permet de calculée l'aire des régions transversales et radiales dans lesquelles les ions diffusent. Quelques lignes de code ont ici été ajoutées pour calculer l'aire de la région transversale dans laquelle le GABA extracellulaire diffuse. Cette région correspond à un anneau qui fait le tour de la cellule. La variable locale 'rgab' se fait assignée la valeur calculée du grand rayon de cet anneau. Finalement, 'areaext' se fait assignée la valeur calculée de l'aire de la région transversale.

6.3.8 Bloc KINETIC state

```
KINETIC state {
    COMPARTMENT areaext {gabo}
    LONGITUDINAL_DIFFUSION DGab*areaext  {gabo}
    ~ gabo <-> bath (1/(taugaba), 1/(taugaba))
    ...
}
```

La partie de code ici ajoutée permet de considérer la diffusion du GABA ainsi que l'échange du GABA avec le bain. La première ligne défini l'aire de la région transversale dans laquelle le GABA diffuse longitudinalement d'un compartiment à l'autre. La seconde ligne calcule cette diffusion longiudinale. Finalement, la dernière ligne considère l'échange avec le bain. Ce dernier schéma cinétique est équivalent à :

$$\mathrm{gabo}' = \frac{1}{\mathrm{taugaba}}(\mathrm{bath} - \mathrm{gabo}) \tag{6.1}$$

et indique une décroissance exponentielle de la concentration de GABA vers 0.

6.4 Modélisation simple de l'échange des ions avec une pipette (*patch clamp*)

6.4.1 Bloc NEURON

```
NEURON {
    ...
    GLOBAL tau, clipip, kipip, naipip
    ...
    RANGE clamp, hco3i0, hco3o0
 }
```

La première ligne ajoutée permet la définition de quatre variables 'GLOBAL'. Ces variables sont 'GLOBAL' et non 'RANGE' puisqu'elles correspondent aux caractéristiques de la pipette *patch clamp* (simplifiée). Comme il n'y a jamais plus d'une pipette, il n'est pas nécessaire d'avoir des caractéristiques différentes pour le mécanisme dans différentes sections du neurone. 'tau' correspond au taux d'échange de concentration avec la pipette. Les trois autres variables correspondent aux concentrations de chlorure, de potassium et de sodium dans la pipette ('clipip', 'kipip' et 'naipip'). À la deuxième ligne ajoutée, la variable 'clamp' a une valeur de 1 si il y a une clampe sur le soma et une valeur de 0 sinon.

6.4.2 Bloc PARAMETER

```
PARAMETER {
    ...
    clamp   = 0
```

```
    clipip  = 8   (mM)
    kipip   = 140 (mM)
    naipip  = 12  (mM)
}
```

'clamp' se fait ici initialisée à 0 (pas de clampe sur le soma). Les concentrations de la pipette se font initialisées une valeur, mais celles-ci sont sans importance puisqu'elles sont en réalité décidées au niveau du script python.

6.4.3 Bloc KINETIC state

```
KINETIC state {
    ...
  if (clamp == 0) {
    cli = cl[0]
    nai = na[0]
    ki  = k[0]
  }
  else {
    ~ cl[0] <-> clipip (1/(tau), 1/(tau))
    ~ na[0] <-> naipip (1/(tau), 1/(tau))
    ~ k[0] <-> kipip (1/(tau), 1/(tau))
    cli = cl[0]
    nai = na[0]
    ki  = k[0]
  }
    ...
}
```

Un bloc conditionnelle a été ajoutée lors de la résolution de la concentration à chaque segment de chaque ion. S'il n'y a pas de clampe sur le soma (clamp = 0), rien ne change pour les concentrations et les concentrations des ions calculées précédemment sont assignées aux valeurs écrites par le script. Sinon, avant de faire cela, trois équations sont ajoutées. Ces équations modélisent de manière simplifiée l'échange des concentrations dans le soma avec la pipette. Comme mentionnée dans la section 6.3.8, cela indique une diminution/augmentation exponentielle des valeurs de concentrations vers les valeurs de concentration de la pipette.

7 *gaghkfinal.mod*

7.1 Script complet

```
COMMENT
-----------------------------------------------------------------------------

  Kinetic model of GABA-A receptors
  =================================

  7-state gating model from Jones and Westbrook (Neuron 15, 181 - 191, 1995)



        D1    D2
              |     |
  C1 -- C2 -- C3
        |     |
              O1    O2

  Or 16-state gating scheme from Haas and MacDonald (J Physiol 514.1, 27 - 45, 1999)


        D2    D3 -- D4
```

```
        |   |
  C1 -- C2 -- C3 -- C4
        |     |     |
       O1     O2    O3
        /\    /\    /\
      C5 C6 C7 C8 C9 C10

Reversal potential Egaba is changing according to [Cl-]i change (due to Cl- influx). Bicarbonate (
    HCO3) flows through the GABAR too, and therefore Egaba is also [HCO3]i/[HCO3]o -dependent.
    igaba = icl + ihco3 (we assume icl and ihco3 to be mutually independent)

Based on gabaA_Cl.mod, modified to use GHK current equation.

Parameters updated from erratum of Haas and MacDonald (J Physiol 514.1, 27 - 45, 1999)

ENDCOMMENT


TITLE detailed GABAergic conductance with changing Cl- concentration

NEURON {
  POINT_PROCESS gaghk

  USEION cl READ cli, clo WRITE icl VALENCE -1
  USEION hco3 READ hco3i, hco3o WRITE ihco3 VALENCE -1
  USEION gab READ gabo VALENCE 0

    RANGE gabo
  RANGE icl, ihco3, igaba

  RANGE f1, f2
  RANGE kon, koff, koff2, k34, k43, alfa1, beta1, alfa2, beta2, alfa3, beta3
  RANGE a1o, a1c, b1o, b1c, a2o, a2c, b2o, b2c, a3o, a3c, b3o, b3c
  RANGE d1, r1, d2, r2, d3, r3, d4, r4
  RANGE C1, C2, C3, C4, C5, C6, C7, C8, C9, C10
  RANGE O1, O2, O3, D1, D2, D3, D4
  RANGE Prel, Pcl, Phco3, Rnumber
  RANGE ecl, ehco3, egaba
  RANGE gcl, ghco3, grel
}

UNITS {
    (mA)    = (milliamp)
    (nA)    = (nanoamp)
  (mV)    = (millivolt)
  (uS)    = (micromho)
  (mM)    = (milli/liter)
  (uM)    = (micro/liter)
  F     = (faraday) (coulombs)
  R     = (k-mole)  (joule/degC)
}

PARAMETER {
  : these must be specified at the hoc level, or through clever use
  : of the INITIAL block

  Prel    = 0.18          : Phco3/Pcl relative permeability
  Rnumber = 10000         : number of GABAARs in the synaptic compartment

  Pcl     = 8e-14 (cm3/s)   : maximum Cl- single channel permeability for GABAAR, https://physoc.
    onlinelibrary.wiley.com/doi/epdf/10.1113/jphysiol.1987.sp016493

  : the value assigned here will have no effect; must be specified at the hoc level
  celsius = 37    (degC)

```

```
  : Rates
  kon     = .007   (/uM /ms)  : binding (.007 Haas)
  koff  = .170   (/ms)    : unbinding (170 Haas)
  koff2   = .300   (/ms)    : unbinding
  alfa1 = 3.1  (/ms)    : closing
  beta1 = .05  (/ms)    : opening
  alfa2 = .280   (/ms)    : closing
  beta2 = 1.8    (/ms)    : opening
  alfa3   = .150   (/ms)    : closing
  beta3   = .076   (/ms)    : opening
  d1      = .013   (/ms)    : slow desensitizing
  r1    = .00049 (/ms)    : resensitizing
  d2    = .960   (/ms)    : fast desensitizing (750 - 1000), 960 (Haas)
  r2    = .022   (/ms)    : resensitizing (15 - 25), 22 (Haas)
  d3    = .008   (/ms)    : intermediate desensitizing
  r3    = .00081 (/ms)    : resensitizing
  d4    = .00075 (/ms)    : slow desenzitizing
  r4    = .00049 (/ms)    : resenzitizing
  k34   = .710   (/ms)
  k43   = .058   (/ms)
  a1o   = 5.1  (/ms)
  a2o   = 5.1  (/ms)
  a1c   = .180   (/ms)
  a2c   = .180   (/ms)
  b1o   = .63    (/ms)
  b2o   = .63    (/ms)
  b1c   = .07    (/ms)
  b2c   = .07    (/ms)
  a3o   = 5.1    (/ms)
  b3o   = .63    (/ms)
  a3c   = .09    (/ms)
  b3c   = .035   (/ms)
}


ASSIGNED {
  v   (mV)    : postsynaptic voltage - we hypothesize that Egaba changes due to increase of [Cl]i

  cli     (mM)
  clo   (mM)
  icl     (nA)    : chloride current
  ecl   (mV)    : equilibrium potential for Cl-

  hco3i   (mM)
  hco3o   (mM)
  ihco3       (nA)    : bicarb current
  ehco3   (mV)    : equilibrium potential for HCO3-

  igaba       (nA)      : total current generated by this mechanism = icl + ihco3
  egaba   (mV)    : reversal potential for GABAR

  gabo    (mM)    : transmitter concentration

    f1    (/ms)     : binding
  f2    (/ms)     : binding

  Phco3     (cm3/s)   : max Phco3 = 0.18 * max Pcl
  gcl     (uS)    : GABA - induced conductance for chloride
  ghco3   (uS)    : GABA - induced conductance for bicarbonate
  grel          : relative conductance
}

STATE {
    : Channel states (all fractions)
  C1    : unbound
```

```
  C2    : single bound
  C3    : double bound
  C4    : double bound
  C5
  C6
  C7
  C8
  C9
  C10
  O1    : open
  O2    : open
  O3    : open
  D1    : desensitized
  D2    : desensitized
  D3    : desensitized
  D4    : desensitized
}

INITIAL {
  :hco3o   = 26    (mM)     : extracellular HCO3- concentration
  :hco3i   = 16    (mM)     : intracellular HCO3- concentration
  gabo    = 0   (mM)
  C1    = 1
  C2    = 0
  C3    = 0
  C4    = 0
  C5    = 0
  C6    = 0
  C7    = 0
  C8    = 0
  C9    = 0
  C10   = 0
  O1    = 0
  O2    = 0
  O3    = 0
  D1    = 0
  D2    = 0
  D3    = 0
  D4    = 0
  icl     = 0
  ihco3   = 0
  igaba   = 0
  Phco3   = Prel * Pcl
}

BREAKPOINT {
  SOLVE kstates METHOD sparse

  icl   = (1e+06)*(O1+O2+O3) * Pcl * Rnumber * ghk(v, cli, clo, -1)          :1e+6 is a factor to
    convert mA to nA
  ihco3 = (1e+06)*(O1+O2+O3) * Phco3 * Rnumber * ghk(v, hco3i, hco3o, -1)
  igaba = icl + ihco3

  egaba = ghkvoltage(cli, clo, hco3i, hco3o)
  gcl = (1e+06)*(O1+O2+O3) * Pcl * Rnumber * conduct(v, cli, clo)   :1e+6 is a factor to convert S
     to uS
  ghco3 = (1e+06)*(O1+O2+O3) * Phco3 * Rnumber *conduct(v, hco3i, hco3o)
  if (gcl>0) {grel = ghco3/gcl} else {grel = 0}
}

KINETIC kstates {
  f1    = 2 * kon * (1e3) * gabo
  f2    = kon * (1e3) * gabo

  ~ C1 <-> C2   (f1,koff)
```

```
  ~ C2 <-> C3   (f2,koff2)
  ~ C2 <-> O1   (beta1,alfa1)
  ~ C3 <-> O2   (beta2,alfa2)
  ~ C2 <-> D1     (d1,r1)
  ~ C3 <-> D2     (d2,r2)

  : Haas and MacDonald kinetics
  ~ C3 <-> C4     (k34,k43)
  ~ C4 <-> O3     (beta3,alfa3)
  ~ C4 <-> D3     (d3,r3)
  ~ D3 <-> D4     (d4,r4)

  ~ O1 <-> C5     (a1c,a1o)
  ~ O1 <-> C6     (b1c,b1o)
  ~ O2 <-> C7     (a2c,a2o)
  ~ O2 <-> C8     (b2c,b2o)
  ~ O3 <-> C9     (a3c,a3o)
  ~ O3 <-> C10    (b3c,b3o)

  CONSERVE C1+C2+C3+C4+C5+C6+C7+C8+C9+C10+O1+O2+O3+D1+D2+D3+D4 = 1
}

FUNCTION ghk(v(mV), ci(mM), co(mM), z)  (millicoul/cm3) {
  LOCAL e, w
        w = v * (.001) * z*F / (R*(celsius+273.15))
        if (fabs(w)>1e-4)
      { e = w / (exp(w)-1) }
        else
      : denominator is small -> Taylor series
            { e = 1-w/2 }
        ghk = - (.001) * z* F * (co-ci*exp(w)) * e
}

FUNCTION ghkvoltage(c1i(mM), c1o(mM), c2i(mM), c2o(mM))  (mV) {
  ghkvoltage = - (1000)*(celsius + 273.15)*R/F*log((c1o + Prel*c2o)/(c1i + Prel*c2i))
}

FUNCTION conduct(v(mV), ci(mM), co(mM))  (millicoul/cm3/mV) {
  LOCAL w
        w = v * (.001) *F / (R*(celsius+273.15))
  conduct = (0.001)*(.001)*F^2/(R*(celsius+273.15))*(ci-(co+ci)*exp(w)+(ci-co)*w*exp(w)+co*(exp(w)
    ^2))/((1-exp(w))^2)
}
```

7.2 Information générale

Ce code provient d'une modification du modèle suivant : *Effects of Chloride accumulation and diffusion on GABAergic transmission* [4]. Celui-ci est disponible sur ModelDB [5]. Ce mécanisme modélise une synapse GABAergique.

7.3 Bloc NEURON

```
NEURON {
  POINT_PROCESS gaghk

  USEION cl READ cli, clo WRITE icl VALENCE -1
  USEION hco3 READ hco3i, hco3o WRITE ihco3 VALENCE -1
  USEION gab READ gabo VALENCE 0

    RANGE gabo
  RANGE icl, ihco3, igaba

  RANGE f1, f2
```

```
  RANGE kon, koff, koff2, k34, k43, alfa1, beta1, alfa2, beta2, alfa3, beta3
  RANGE a1o, a1c, b1o, b1c, a2o, a2c, b2o, b2c, a3o, a3c, b3o, b3c
  RANGE d1, r1, d2, r2, d3, r3, d4, r4
  RANGE C1, C2, C3, C4, C5, C6, C7, C8, C9, C10
  RANGE O1, O2, O3, D1, D2, D3, D4
  RANGE Prel, Pcl, Phco3, Rnumber
  RANGE ecl, ehco3, egaba
  RANGE gcl, ghco3, grel
}
```

la première ligne du bloc indique que le mécanisme est un *point process* nommé gaghk. Les deux lignes suivantes indiquent l'utilisation des ions chlorure et HCO3-. Les concentrations intracellulaires et extracellulaires des deux ions sont obtenues et un courant est généré pour chacun. La ligne suivante est un ajout au code original et indique l'utilisation de l'"ion" GABA dont la concentration extracellulaire est obtenue. De nombreuses variables 'RANGE' sont ensuite initialisées dont la concentration extracellulaire de GABA ('gabo') et les courants générées par le mécanismes ('icl', 'ihco3' et 'igaba' = 'icl' + 'ihco3'). Le rôle des autres variables sera expliqué dans les sections suivantes.

7.4 Bloc PARAMETER

```
PARAMETER {
  : these must be specified at the hoc level, or through clever use
  : of the INITIAL block

  Prel    = 0.18          : Phco3/Pcl relative permeability
  Rnumber = 10000         : number of GABAARs in the synaptic compartment

  Pcl     = 8e-14 (cm3/s)   : maximum Cl- single channel permeability for GABAAR, https://physoc.
    onlinelibrary.wiley.com/doi/epdf/10.1113/jphysiol.1987.sp016493

  : the value assigned here will have no effect; must be specified at the hoc level
  celsius = 37    (degC)

  : Rates
  kon     = .007   (/uM /ms)  : binding (.007 Haas)
  koff  = .170   (/ms)    : unbinding (170 Haas)
  koff2   = .300   (/ms)    : unbinding
  alfa1 = 3.1  (/ms)    : closing
  beta1 = .05  (/ms)    : opening
  alfa2 = .280   (/ms)    : closing
  beta2 = 1.8    (/ms)    : opening
  alfa3   = .150   (/ms)    : closing
  beta3   = .076   (/ms)    : opening
  d1      = .013   (/ms)    : slow desensitizing
  r1    = .00049 (/ms)    : resensitizing
  d2    = .960   (/ms)    : fast desensitizing (750 - 1000), 960 (Haas)
  r2    = .022   (/ms)    : resensitizing (15 - 25), 22 (Haas)
  d3    = .008   (/ms)    : intermediate desensitizing
  r3    = .00081 (/ms)    : resensitizing
  d4    = .00075 (/ms)    : slow desenzitizing
  r4    = .00049 (/ms)    : resenzitizing
  k34   = .710   (/ms)
  k43   = .058   (/ms)
  a1o   = 5.1  (/ms)
  a2o   = 5.1  (/ms)
  a1c   = .180   (/ms)
  a2c   = .180   (/ms)
  b1o   = .63    (/ms)
  b2o   = .63    (/ms)
  b1c   = .07    (/ms)
  b2c   = .07    (/ms)
  a3o   = 5.1    (/ms)
  b3o   = .63    (/ms)
  a3c   = .09    (/ms)
```

```
  b3c    = .035    (/ms)
}
```

La perméabilité maximale de l'ion chlorure pour un seul canal de GABA correspond au paramètre 'Pcl'. La perméabilité relative (P_{rel}/P_{cl}) pour le HCO3- correspond à 'Prel'. Le nombre de récepteur GABA dans une synapse correspond au paramètre 'R-number' dont la valeur initialisée ici est sans importance puisque celle-ci est décidé au niveau du script python. Le paramètre suivant, 'celsius', correspond à la température. Elle aussi est décidé au niveau du code python. Cette variable est utilisée plus loin dans les calculs de conductances et de courants. Tous les paramètres suivants sont associés aux probabilités de transitions d'un état à l'autre dans le schéma cinétique des récepteurs GABA-A (ouvert, fermé et désensibilisé).

7.5 Bloc ASSIGNED

```
ASSIGNED {
  v    (mV)     : postsynaptic voltage - we hypothesize that Egaba changes due to increase of [Cl]i

  cli      (mM)
  clo    (mM)
  icl      (nA)     : chloride current
  ecl    (mV)     : equilibrium potential for Cl-

  hco3i    (mM)
  hco3o    (mM)
  ihco3        (nA)     : bicarb current
  ehco3    (mV)     : equilibrium potential for HCO3-

  igaba        (nA)       : total current generated by this mechanism = icl + ihco3
  egaba    (mV)     : reversal potential for GABAR

  gabo     (mM)     : transmitter concentration

    f1     (/ms)      : binding
  f2     (/ms)      : binding

  Phco3      (cm3/s)    : max Phco3 = 0.18 * max Pcl
  gcl      (uS)     : GABA - induced conductance for chloride
  ghco3    (uS)     : GABA - induced conductance for bicarbonate
  grel           : relative conductance
}
```

Le potentiel de membrane ('v'), les concentrations intracellulaires et extracellulaires du chlorure et du HCO3- ('cli', 'clo', 'hco3i' et 'hco3o'), les potentiels réversibles du chlorure et du HCO3- ('ecl' et 'ehco3') et la concentration extracellulaire de GABA ('gabo') sont toutes des variables utilisées dans des calculs plus loin. Les courants ('icl', 'ihco3' et 'igaba') ainsi que le potentiel réversible pour les récepteurs GABAR ('egaba') sont des variables se faisant assigner une valeur calculée.

'f1' et 'f2' correspondent à des taux de changement dans le schéma cinétique des récepteurs GABA-A qui dépendent de la concentration extracellulaire de GABA et dont la valeur est calculée. 'Phco3' est la perméabilité de l'ion HCO3- pour un seul canal GABA. Cette variable se fait assigner la valeur $\mathrm{Phco3} = (\mathrm{Prel})(\mathrm{Pcl})$ dans le bloc 'INITIAL' et est utilisée dans des calculs plus loin. Finalement, les conductances des canaux GABAR pour le chlorure et le HCO3- ainsi que la conductance relative sont calculées ('gcl', 'ghco3' et 'grel').

7.6 Bloc STATE

```
STATE {
    : Channel states (all fractions)
  C1     : unbound
  C2     : single bound
  C3     : double bound
  C4     : double bound
  C5
```

```
  C6
  C7
  C8
  C9
  C10
  O1    : open
  O2    : open
  O3    : open
  D1    : desensitized
  D2    : desensitized
  D3    : desensitized
  D4    : desensitized
}
```

Toutes les variables ici correspondent à l'état des canaux GABA. Les variables 'Cx' correspondent aux différents états non ouverts. Les variables 'Ox' correspondent aux états ouverts. Les variables 'Dx' correspondent aux états désensibilisés. Chaque variable a une valeur fractionnaire de 0 à 1 et la somme de toutes les variables est égale à 1 (100% des canaux).

7.7 Bloc INITIAL

```
INITIAL {
  :hco3o   = 26    (mM)     : extracellular HCO3- concentration
  :hco3i   = 16    (mM)     : intracellular HCO3- concentration
  gabo    = 0   (mM)
  C1    = 1
  C2    = 0
  C3    = 0
  C4    = 0
  C5    = 0
  C6    = 0
  C7    = 0
  C8    = 0
  C9    = 0
  C10   = 0
  O1    = 0
  O2    = 0
  O3    = 0
  D1    = 0
  D2    = 0
  D3    = 0
  D4    = 0
  icl     = 0
  ihco3   = 0
  igaba   = 0
  Phco3   = Prel * Pcl
}
```

La concentration initiale de GABA extracellulaire est mise à 0. Tous les canaux partent dans l'état 'C1' fermé. C'est pour cette raison que cette seule variable d'état a une valeur non nulle (100% des canaux dans l'état C1). Les courants générés par le mécanisme sont également initialisés à 0 et la conductance du HCO3- est calculée ('Phco3').

7.8 Bloc BREAKPOINT

```
BREAKPOINT {
  SOLVE kstates METHOD sparse

  icl   = (1e+06)*(O1+O2+O3) * Pcl * Rnumber * ghk(v, cli, clo, -1)          :1e+6 is a factor to
    convert mA to nA
  ihco3 = (1e+06)*(O1+O2+O3) * Phco3 * Rnumber * ghk(v, hco3i, hco3o, -1)
  igaba = icl + ihco3

```

```
  egaba = ghkvoltage(cli, clo, hco3i, hco3o)
  gcl = (1e+06)*(O1+O2+O3) * Pcl * Rnumber * conduct(v, cli, clo)   :1e+6 is a factor to convert S
     to uS
  ghco3 = (1e+06)*(O1+O2+O3) * Phco3 * Rnumber *conduct(v, hco3i, hco3o)
  if (gcl>0) {grel = ghco3/gcl} else {grel = 0}
}
```

La première ligne indique à NEURON de résoudre les équation cinétiques du bloc 'KINETIC kstates' par la méthode 'sparse'.

Ensuite en utilisant le formalisme d'équations Goldman–Hodgkin–Katz, les courants de chlorure et de HCO3- ('icl' et 'ihco3') sont calculés en considérant également le nombre de canaux ouverts et le nombre de récepteurs. Le courant total généré par la synapse est aussi calculé ('igaba'). La fonction appelée lors de ces calculs (*ghk*) permet de simplifier l'écriture et d'effectuer une série de taylor lorsque le dénominateur approche une valeur de 0.

Par la suite, le potentiel réversible du GABAR est calculé grâce à fonction *ghkvolatge* qui permet de simplifier l'écriture. Les conductances du chlorure et du HCO3- sont aussi calculés en considérant le nombre de canaux ouverts et le nombre de récepteur et en utilisant la fonction *conduct* qui permet la simplification de l'écriture. Finalement, la conductance relative est calculée. Un bloc conditionnel est ajouté afin d'éviter une division par 0 dans le cas où 'gcl' est nul.

7.9 Bloc KINETIC kstates

```
KINETIC kstates {
  f1    = 2 * kon * (1e3) * gabo
  f2    = kon * (1e3) * gabo

  ~ C1 <-> C2   (f1,koff)
  ~ C2 <-> C3   (f2,koff2)
  ~ C2 <-> O1   (beta1,alfa1)
  ~ C3 <-> O2   (beta2,alfa2)
  ~ C2 <-> D1     (d1,r1)
  ~ C3 <-> D2     (d2,r2)

  : Haas and MacDonald kinetics
  ~ C3 <-> C4     (k34,k43)
  ~ C4 <-> O3     (beta3,alfa3)
  ~ C4 <-> D3     (d3,r3)
  ~ D3 <-> D4     (d4,r4)

  ~ O1 <-> C5     (a1c,a1o)
  ~ O1 <-> C6     (b1c,b1o)
  ~ O2 <-> C7     (a2c,a2o)
  ~ O2 <-> C8     (b2c,b2o)
  ~ O3 <-> C9     (a3c,a3o)
  ~ O3 <-> C10    (b3c,b3o)

  CONSERVE C1+C2+C3+C4+C5+C6+C7+C8+C9+C10+O1+O2+O3+D1+D2+D3+D4 = 1
}
```

Aux deux premières lignes du bloc, les taux de transitions qui sont dépendant de la concentration de GABA extracellulaire sont calculés. Tout ce qui suit correspond au différentes équations cinétiques associés au schéma cinétique du GABAR. Entre parenthèse à côté de chaque équation se trouve les probabilités de transition d'un état vers l'autre. La dernière ligne débutant par 'CONSERVE' assure que la somme des états est toujours conservée à 1.

8 *hh_rat.mod*

8.1 Script complet

```
TITLE hh.mod   squid sodium, potassium, and leak channels

COMMENT
 This is the original Hodgkin-Huxley treatment for the set of sodium,
  potassium, and leakage channels found in the squid giant axon membrane.
  ("A quantitative description of membrane current and its application
  conduction and excitation in nerve" J.Physiol. (Lond.) 117:500-544 (1952).)
 Membrane voltage is in absolute mV and has been reversed in polarity
  from the original HH convention and shifted to reflect a resting potential
  of -65 mV.
 Remember to set a squid-appropriate temperature
 (e.g. in HOC: "celsius=6.3" or in Python: "h.celsius=6.3").
 See squid.hoc for an example of a simulation using this model.
 SW Jaslove  6 March, 1992
ENDCOMMENT

UNITS {
        (mA) = (milliamp)
        (mV) = (millivolt)
        (S) = (siemens)
}

? interface
NEURON {
        SUFFIX hhrat
        REPRESENTS NCIT:C17145   : sodium channel
        REPRESENTS NCIT:C17008   : potassium channel
        USEION na READ ena WRITE ina REPRESENTS CHEBI:29101
        USEION k READ ek WRITE ik REPRESENTS CHEBI:29103
        NONSPECIFIC_CURRENT il
        RANGE gnabar, gkbar, gl, el, gna, gk, ik, ina, il
        : `GLOBAL minf` will be replaced with `RANGE minf` if CoreNEURON enabled
        GLOBAL minf, hinf, ninf, mtau, htau, ntau
        THREADSAFE : assigned GLOBALs will be per thread
}

PARAMETER {
        gnabar = .12 (S/cm2)    <0,1e9> : .040
        gkbar = .036 (S/cm2)    <0,1e9> : .035
        gl = .0003 (S/cm2)      <0,1e9>
        el = -54.3 (mV)
}

STATE {
        m
        h
        n
}

ASSIGNED {
        v (mV)
        celsius (degC)
        ena (mV)
        ek (mV)

        gna (S/cm2)
        gk (S/cm2)
        ina (mA/cm2)
        ik (mA/cm2)
        il (mA/cm2)
        minf hinf ninf
        mtau (ms)
        htau (ms)
        ntau (ms)
}
```

```
66
67 ? currents
68 BREAKPOINT {
69         SOLVE states METHOD cnexp
70         gna = gnabar*m*m*m*h
71         ina = gna*(v - ena)
72         gk = gkbar*n*n*n*n
73         ik = gk*(v - ek)
74         il = gl*(v - el)
75 }
76
77
78 INITIAL {
79         rates(v)
80         m = minf
81         h = hinf
82         n = ninf
83 }
84
85 ? states
86 DERIVATIVE states {
87         rates(v)
88         m' =  (minf-m)/mtau
89         h' = (hinf-h)/htau
90         n' = (ninf-n)/ntau
91 }
92
93
94 ? rates
95 PROCEDURE rates(v(mV)) {  :Computes rate and other constants at current v.
96                         :Call once from HOC to initialize inf at resting v.
97         LOCAL  alpha, beta, sum, q10
98         : `TABLE minf` will be replaced with `:TABLE minf` if CoreNEURON enabled
99         TABLE minf, mtau, hinf, htau, ninf, ntau DEPEND celsius FROM -100 TO 100 WITH 200
100
101 UNITSOFF
102         q10 = 3^((celsius - 23)/10)
103
104                 :"m" sodium activation system
105         alpha = -0.182 * vtrap(-(v+35), 9)
106         beta =  -0.124 * vtrap((v+35),9)
107         sum = alpha + beta
108         mtau = 1/(q10*sum)
109         minf = alpha/sum
110
111                 :"h" sodium inactivation system
112         alpha = 0.25 * exp(-(v+90)/12)
113         beta = 0.25 * exp((v+62)/6 - (v+90)/12)
114         sum = alpha + beta
115         htau = 1/(q10*sum)
116         hinf = alpha/sum
117
118                 :"n" potassium activation system
119         alpha = -0.02 * vtrap(-(v-25), 9)
120         beta = -0.002 * vtrap((v-25),9)
121         sum = alpha + beta
122         ntau = 1/(q10*sum)
123         ninf = alpha/sum
124 }
125
126 FUNCTION vtrap(x,y) {  :Traps for 0 in denominator of rate eqns.
127         if (fabs(x/y) < 1e-6) {
128                 vtrap = -y + x/2
129         }else{
130                 vtrap = x/(1-exp(x/y))
```

```
        }
}

UNITSON
```

8.2 Information générale

Ce fichier mod a été modifié et provient possiblement du modèle *Action potential-evoked Na+ influx are similar in axon and soma* [6] disponible sur ModelDB [7]. Ce mécanisme modélise les canaux Hodgkin Huxley de potassium et de sodium.

8.3 Bloc NEURON

```
NEURON {
        SUFFIX hhrat
        REPRESENTS NCIT:C17145   : sodium channel
        REPRESENTS NCIT:C17008   : potassium channel
        USEION na READ ena WRITE ina REPRESENTS CHEBI:29101
        USEION k READ ek WRITE ik REPRESENTS CHEBI:29103
        NONSPECIFIC_CURRENT il
        RANGE gnabar, gkbar, gl, el, gna, gk, ik, ina, il
        : `GLOBAL minf` will be replaced with `RANGE minf` if CoreNEURON enabled
        GLOBAL minf, hinf, ninf, mtau, htau, ntau
        THREADSAFE : assigned GLOBALs will be per thread
}
```

Les deux premières lignes du bloc indique de quels canaux il s'agit avec précision. Les deux lignes suivantes indiquent l'utilisation des ions sodium et potassium dont le potentiel réversible est obtenu. Un courant est également généré pour ces deux ions par le mécanisme. La troisième ligne défini un courant qui n'est spécifique à aucun ion.

Des variables 'RANGE' sont ensuite définies. 'gnabar' et 'gnabar' correspondent respectivement aux conductances maximales des canaux de sodium et de potassium. Les variables 'gl', 'gna' et 'gk' correspondent aux conductances des canaux de fuite, de sodium et de potassium. 'el' est le potentiel réversible des canaux de fuites. Finalement, les variables globales restantes correspondent aux valeurs des x_∞ et des τ_x typiques des équations d'Hudgkin Huxley.

8.4 Bloc PARAMETER

```
PARAMETER {
        gnabar = .12 (S/cm2)     <0,1e9> : .040
        gkbar = .036 (S/cm2)     <0,1e9> : .035
        gl = .0003 (S/cm2)       <0,1e9>
        el = -54.3 (mV)
}
```

Initialisation des paramètres dont les conductances maximales, la conductance des canaux de fuite et le potentiel réversible des canaux de fuites. Les valeurs dans '<...>' correspondent aux bornes de l'éventail de valeurs possibles pour le paramètre.

8.5 Bloc STATE

```
STATE {
        m
        h
        n
}
```

Les variables résolues. 'm' correspond à l'activation des canaux de sodium, 'h' à l'inactivation des canaux de sodium et 'n' à l'activation des canaux de potassium.

8.6 Bloc ASSIGNED

```
ASSIGNED {
        v (mV)
        celsius (degC)
        ena (mV)
        ek (mV)

        gna (S/cm2)
        gk (S/cm2)
        ina (mA/cm2)
        ik (mA/cm2)
        il (mA/cm2)
        minf hinf ninf
        mtau (ms)
        htau (ms)
        ntau (ms)
}
```

Les quatre premières variables sont utilisées plus loin dans les calculs. Les conductances et les courants ('gna', 'gk', 'ina', 'ik' et 'il') se font assigner une valeur calculée. Les dernières variables sont calculées et dépendent des variables 'STATE' résolues. Elles sont aussi ensuite utilisées dans d'autres calculs.

8.7 Bloc BREAKPOINT

```
BREAKPOINT {
        SOLVE states METHOD cnexp
        gna = gnabar*m*m*m*h
        ina = gna*(v - ena)
        gk = gkbar*n*n*n*n
        ik = gk*(v - ek)
        il = gl*(v - el)
}
```

La première ligne dit à NEURON de résoudre les équations du bloc 'DERIVATIVE states' grâce à la méthode 'cnexp'. Après avoir fait cela, les valeurs des variables 'm', 'h' et 'n' sont obtenus. Cela permet le calcul des conductances ('gna' et 'gk') qui permettent à leur tour le calcul des courants ('ina' et 'ik'). Le courant de fuite est finalement calculé. Dans le modèle, 'gl' est habituellement mis à 0 et il n'y a donc aucun courant de fuite.

8.8 Bloc INITIAL

```
INITIAL {
        rates(v)
        m = minf
        h = hinf
        n = ninf
}
```

Fait d'abord appel à la fonction *rates* qui prend en entrée le potentiel de membrane et qui calcule les x_∞ et les τ_x. Les valeurs de 'm', 'h' et 'n' sont initialisées à la valeur des x_∞.

8.9 Bloc DERIVATIVE states

```
DERIVATIVE states {
        rates(v)
        m' =  (minf-m)/mtau
        h' = (hinf-h)/htau
        n' = (ninf-n)/ntau
}
```

la première ligne montre que le bloc fait d'abord appel à la fonction *rates* qui prend en entrée le potentiel de membrane et qui calcule les x_∞ et les τ_x. Les trois prochaines lignes montrent les équations différentielles typiques du modèle Hudgkin Hudxley qui sont résolues.

8.10 Bloc PROCEDURE rates

```
PROCEDURE rates(v(mV)) {  :Computes rate and other constants at current v.
                      :Call once from HOC to initialize inf at resting v.
        LOCAL  alpha, beta, sum, q10
        : `TABLE minf` will be replaced with `:TABLE minf` if CoreNEURON enabled
        TABLE minf, mtau, hinf, htau, ninf, ntau DEPEND celsius FROM -100 TO 100 WITH 200

UNITSOFF
        q10 = 3^((celsius - 23)/10)

                :"m" sodium activation system
        alpha =  -0.182 * vtrap(-(v+35), 9)
        beta =  -0.124 * vtrap((v+35),9)
        sum = alpha + beta
        mtau = 1/(q10*sum)
        minf = alpha/sum

                :"h" sodium inactivation system
        alpha = 0.25 * exp(-(v+90)/12)
        beta = 0.25 * exp((v+62)/6 - (v+90)/12)
        sum = alpha + beta
        htau = 1/(q10*sum)
        hinf = alpha/sum

                :"n" potassium activation system
        alpha = -0.02 * vtrap(-(v-25), 9)
        beta = -0.002 * vtrap((v-25),9)
        sum = alpha + beta
        ntau = 1/(q10*sum)
        ninf = alpha/sum
}
```

Comme mentionné précédemment, c'est là que les variables des équations dans le bloc 'DERIVATIVE' sont calculées. 'q10' est une variable locale qui permet de considérer la température lors des calculs. La fonction *vtrap* appelée à plusieurs reprises dans le bloc permet de simplifier l'écriture et d'effectuer une série de Taylor lorsque le dénominateur dans les calculs approchent de 0.

Références

[1] Vadym Gnatkovsky Damiano Gentiletti Piotr Suffczynski et Marco de Curtis. « Changes of Ionic Concentrations During Seizure Transitions - A Modeling Study ». In : *international Journal of Neural Systems* 27.4 (2017). doi : 10.1142/S0129065717500046.

[2] Gnatkovsky V Gentiletti D Suffczynski P et de Curtis M. « Changes of Ionic Concentrations During Seizure Transitions ». In : *ModelDB* (2016).

[3] M L Hines et N T Carnevales. « Expanding NEURON's Repertoire of Mechanisms with NMODL ». In : *Neural Computation* (1999).

[4] Boris S Gutkin Peter Jedlicka Thomas Deller et Kurt H Backus. « Activity-Dependent Intracellular Chloride Accumulation and Diffusion Controls GABAA Receptor-Mediated Synaptic Transmission ». In : *Hippocampus* 21.6 (2011). doi : 10.1002/hipo.20804.

[5] Boris S Gutkin Peter Jedlicka Thomas Deller et Kurt H Backus. « Effects of Chloride accumulation and diffusion on GABAergic transmission ». In : *ModelDB* (2011).

[6] Michael J Gutnick Ilya A Fleidervish Nechama Lasser-Ross et William N Ross. « Na+ imaging reveals little difference in action potential–evoked Na+ influx between axon and soma ». In : *Nature Neuroscience* 13 (2010). doi : https://doi.org/10.1038/nn.2574.

[7] Michael J Gutnick Ilya A Fleidervish Nechama Lasser-Ross et William N Ross. « Action potential-evoked Na+ influx are similar in axon and soma ». In : *ModelDB* (2010).