



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

PROJET DE SEMESTRE MASTER CMCS
(PROF. ALFIO QUARTERONI)
Année 2011-2012 – Semestre de printemps

Modèles mathématiques pour l'activité électrique du cœur et des oreillettes

Auteur :

Marie DUPRAZ (178516)

Supervisé par :

Dr. Ricardo RUIZ BAIER

Dernière modification le 14 juin 2012

Résumé

Ce projet présente les bases physiques nécessaires à la modélisation de l'activité électrique dans le cœur et les oreillettes ainsi que plusieurs modèles mathématiques développés à cet effet. On donne les spécificités de chacun de ceux-ci en terme de rendu du potentiel de la membrane cellulaire. On étudie ensuite plus en détails le modèle de Courtemanche-Ramirez-Nattel [CRN98] pour le potentiel dans les oreillettes dont on valide le code implémenté en LifeV [Lif].

La seconde partie du travail présente un problème de contrôle optimal : on dérive le modèle théorique pour les équations monodomaines avec le modèle de Rogers-McCulloch [RM94], puis on applique ces résultats à la défibrillation d'un domaine excité.

Table des matières

| | | |
|-------------------|---|-----------|
| 1 | Introduction | 5 |
| I | Présentation et analyse de modèles numériques pour l'activité électrique cardiaque | 6 |
| 2 | Modèles mathématiques | 6 |
| 2.1 | Corps conducteur | 6 |
| 2.2 | Modélisation de l'activité électrique dans les tissus cardiaques | 7 |
| 2.2.1 | Définition des tissus excitables | 7 |
| 2.2.2 | Modèle bidomaine | 8 |
| 2.2.3 | Modèle monodomaine | 10 |
| 2.3 | Modélisation du courant ionique | 11 |
| 2.3.1 | Modèles phénoménologiques | 11 |
| 2.3.2 | Modèles physiologiques | 13 |
| 3 | Discrétisation du modèle | 21 |
| 3.1 | Formulation variationnelle | 21 |
| 3.2 | Semi-discrétisation en espace | 22 |
| 3.3 | Discrétisation totale | 23 |
| 4 | Résultats numériques | 24 |
| 4.1 | Modèle RM | 24 |
| 4.2 | Validation du code CRN | 25 |
| 4.3 | Comparaison entre les modèles RM et CRN | 28 |
| 4.4 | Instigation d'ondes spirales | 28 |
| II | Contrôle optimal en électrophysiologie cardiaque | 33 |
| 5 | Problème inverse en électrocardiographie | 33 |
| 6 | Contrôle optimal pour les équations monodomaines | 33 |
| 6.1 | Présentation du problème | 33 |
| 6.2 | Dérivation du problème de contrôle optimal | 34 |
| 6.2.1 | Lagrangienne du système | 34 |
| 6.2.2 | Problème adjoint | 34 |
| 6.2.3 | Condition d'optimalité | 35 |
| 7 | Résolution numérique du problème de contrôle optimal | 35 |
| 7.1 | Boucle d'optimisation | 36 |
| 7.2 | Comportement physique du problème | 36 |
| 8 | Conclusion | 41 |
| Références | | 42 |
| Annexes | | 44 |

| | |
|--|-----------|
| A Modèle RM | 44 |
| A.1 Code FreeFEM pour la résolution du modèle RM | 44 |
| B Modèle CRN | 45 |
| B.1 Code Matlab pour la résolution du modèle ionique | 45 |
| B.2 Corrections du modèle CRN de LifeV | 53 |

1 Introduction

Ce travail consiste en deux aspects principaux : l'étude de la physique et des différents modèles mathématiques qui permettent la modélisation de l'activité électrique du cœur et des oreillettes, puis la dérivation d'un problème de contrôle optimal pour les équations monodomaines.

Le but de ce travail est avant tout d'étudier le modèle de Courtemanche-Ramirez-Nattel (CRN) pour la propagation du potentiel électrique dans les oreillettes ainsi que de valider le code LifeV pour ce modèle.

Dans un premier temps, on pose les bases physiques nécessaires à la compréhension de l'activité électrique cardiaque. Les spécificités en matière d'excitabilité et de conductivité des tissus cardiaques nous permettent de dériver les équations bidomaines et monodomaines pour l'évolution du potentiel électrique dans les tissus du cœur. On constate que ce potentiel dépend du courant ionique résultant d'échange entre les cellules cardiaques. Celui-ci peut être décrit selon des caractéristiques phénoménologiques ou physiologiques. Dans chacun des ces cas, plusieurs modèles ont été développés pour décrire le courant ionique, parmi lesquels on présente les plus importants. On s'attarde en particulier sur le modèle CRN. Celui-ci diffère des précédents car il modélise spécifiquement le potentiel électrique dans les oreillettes.

Puis, on s'intéresse à l'implémentation numérique des modèles que l'on a exposés précédemment. On se concentre ici sur les modèles de Rogers-McCulloch et Courtemanche-Ramirez-Nattel. Pour le premier, on effectue d'abord des comparaisons entre le comportement en deux et trois dimensions dans le but de mettre en parallèle les implémentations en FreeFEM++ et LifeV. Ceci permet de valider les codes pour le modèle en question. On peut ainsi l'utiliser pour effectuer des simulations sur l'oreillette gauche et étudier le comportement des ondes spirales.

Un aspect clé du travail est la validation du code LifeV pour le modèle CRN. Pour ce faire on compare nos simulations avec celle de [GG08] que l'on utilise comme référence.

Enfin, on peut comparer les deux modèles numériques implémentés. L'un se basant sur le comportement macroscopique des cellules cardiaques et l'autre sur le comportement microscopique, il apparaît que les potentiels ainsi modélisés présentent des caractéristiques différentes. Si le premier modèle séduit par sa simplicité, on remarque qu'il ne traduit pas toutes les subtilités des phases du potentiel d'action.

Dans un second temps, on s'intéresse à un problème de contrôle optimal pour les équations monodomaines. La résolution de ce type de problème est cruciale pour le traitement des problèmes inverses. Or, dans le domaine de l'électrophysiologie cardiaque, ceux-ci occupent une place centrale. En effet, ils veulent notamment permettre la reconstruction du potentiel cardiaque à grâce à des mesures à la surface du torse – ce qui est le mode de diagnostic classique, grâce à l'électrocardiogramme.

En ce sens, le problème que l'on présente cherche à déterminer l'amplitude optimale d'un stimulus de défibrillation. Pour souci de simplicité, on se place dans le cadre des équations monodomaines et on utilise le modèle ionique de Rogers-McCulloch. Le but est d'annuler l'effet d'une stimulation initiale par deux stimuli d'amplitude optimale.

Finalement, on présente à la fin de chaque partie l'implémentation numérique de tous ces modèles et méthodes. Pour ce faire, on utilise différents programmes : Matlab [MAT10] pour les modèles ioniques zéro-dimensionnel, FreeFEM++ [Hec09] pour les simulations 2D et LifeV pour les simulations 3D. Tous les codes sont présentés et expliqués en fin de travail.

Première partie

Présentation et analyse de modèles numériques pour l'activité électrique cardiaque

2 Modèles mathématiques

La compréhension de l'activité électrique du cœur fait intervenir certains concepts physiques clés. On commence ainsi par exposer la base nécessaire à la construction des modèles mathématiques pour le potentiel d'action.

On introduit ici une façon de modéliser l'activité électrique du cœur. Pour cela, on se place d'abord dans le cadre macroscopique et on décrit l'activité électrique du cœur par rapport à la place quelle occupe dans le corps. On se concentre ensuite sur les tissus et, plus précisément, les cellules cardiaques. Ceci nous permet d'établir les équations bidomaines, puis de déduire le modèle monodomaine.

Puis, on remarque que le potentiel dépend du courant ionique qui passe au travers des parois cellulaires. Ainsi, l'établissement d'un modèle complet demande de connaître ce courant, ce qui amène à l'étude d'un modèle cellulaire. Celui-ci régit l'évolution temporelle du potentiel dans une cellule. Il apparaît alors que ce dernier peut s'établir sur la base de considérations phénoménologiques ou physiologiques ; on présente les modèles les plus connus pour chacun de ces cas.

Enfin, on présente plus en détails le modèle de Courtemanche-Ramirez-Nattel [CRN98] pour le potentiel électrique dans les oreillettes. Ce dernier est également au centre des simulations numériques effectuées en seconde partie.

2.1 Corps conducteur

Le corps humain n'est rien d'autre qu'un groupe de cellules reliées entre elles par des mécanismes de couplage relatifs aux tissus auxquels elles appartiennent. De cette manière, si l'on veut modéliser l'activité du corps, on peut modéliser l'activité de chaque cellule puis grouper ces informations. Cependant, cela nécessite le stockage de beaucoup trop d'informations à cause du nombre très élevé de cellules. Cette stratégie n'est donc pas viable pour des études au niveau macroscopiques.

Pour palier ce problème, on adopte une méthode basée sur des moyennes volumiques. En d'autres termes, pour une quantité $q(P)$ mesurée en un point P du corps, on considère la moyenne de cette quantité sur un volume choisi autour de P . Si l'on travaille au niveau cellulaire, on doit avoir

$$\text{Volume(cellule } c) \ll \text{Volume(choisi autour de } c) \ll \text{Volume(corps)}.$$

Cette approche permet de modéliser le corps comme un volume conducteur. On donne maintenant la dérivation mathématique de ce premier modèle.

Tout d'abord, on rappelle que la relation de Maxwell entre un champ électrique E et un champ magnétique B est donnée par

$$\nabla \times E + \frac{\partial B}{\partial t} = 0.$$

Dans le cadre de la théorie du volume conducteur, les répercussions de l'activité électrique du cœur sur le champ électrique est assez lente. De cette manière, on peut supposer que le champ E est quasi-statique, c'est-à-dire

$$\nabla \times E = 0$$

et ainsi il dérive du potentiel électrique u , autrement dit

$$E = -\nabla u.$$

Toutes les autres quantités sont issues de moyennes volumiques. On sait que le courant J dans un conducteur est donné par

$$J = ME = -M\nabla u, \quad (2.1)$$

où M est un tenseur décrivant la *conductivité* du milieu.

On suppose encore qu'il n'y a ni source, ni perte de courant et qu'il n'y pas création de charge. Ceci s'exprime aussi

$$\int_S J \cdot n dS = 0, \quad \forall S,$$

ce qui, par le théorème de la divergence, est équivalent à

$$-\int_V \nabla \cdot J dV = 0, \quad \forall V,$$

où V est un volume contenu dans le corps, S sa surface et n sa normale extérieure. Comme cette propriété est valable pour tout sous-volume inclus dans le corps considéré, on en déduit

$$\nabla \cdot J = -\nabla \cdot (M\nabla u) = 0.$$

Dans le but de modéliser l'activité cardiaque, il nous faut modifier légèrement ce modèle car celle-ci est génératrice de courant. Cette source de courant est vue comme un ou plusieurs dipôles électriques.

Comme le corps est entouré d'air qui est un milieu isolant, la composante du courant perpendulaire au corps est nulle à la surface de celui-ci, c'est-à-dire $J \cdot n = 0$. Finalement, on obtient le système d'équations aux dérivées partielles pour le potentiel électrique u dans le corps $\Omega \subset \mathbb{R}^d$ donnée par :

$$\begin{cases} \nabla \cdot (M\nabla u) = f, & \text{dans } \Omega, \\ n \cdot M\nabla u = 0, & \text{sur } \partial\Omega, \end{cases}$$

où f est un terme de source qui satisfait $\int_\Omega f d\Omega = 0$.

2.2 Modélisation de l'activité électrique dans les tissus cardiaques

2.2.1 Définition des tissus excitables

Les tissus cardiaques sont considérés comme des tissus excitables dans le sens où ils sont constitués de cellules qui répondent activement aux stimuli électriques. De plus, les cellules cardiaques transmettent leur excitation électrique aux cellules environnantes. De cette manière, la stimulation d'une partie de cœur se propage dans tout le cœur.

2.2.2 Modèle bidomaine

Tout comme le modèle présenté en 2.1, le modèle bidomaine est basé sur l'utilisation de moyennes volumiques. On s'intéresse ici à la différence de potentiel entre l'intérieur et l'extérieur de la cellule le long de la membrane cellulaire. Le domaine d'étude est ainsi séparé en deux : l'intérieur et l'extérieur de la cellule.

On suppose que chacun de ces sous-domaines est continu et rempli tout l'espace. Cette hypothèse de continuité induit l'existence de jonctions communicantes entre l'intérieur des différentes cellules. Il s'agit de canaux perméables à certains ions particuliers. Ce mécanisme est expliquer de manière plus détaillée à la section 2.3.2.

On nomme u_i et u_e les potentiels intra (resp. extra) cellulaires. Il s'agit de quantités issues de moyenne sur des petits volumes. On suppose encore que chaque point x du corps que l'on considère possède un potentiel $u_i(x)$ et un potentiel $u_e(x)$.

La membrane cellulaire est considérée comme un isolant électrique ce qui permet d'avoir une différence de potentiel entre l'intérieur et l'extérieur de la cellule. Ceci autorise également une séparation des charges entre les deux domaines. On appelle la différence de potentiel le long de la membrane cellulaire, $v = u_i - u_e$ le « potentiel trans-membrane ».

Comme en (2.1), on peut supposer que les courants sont quasi-statiques. Ainsi, on obtient les courants intra (resp. extra) cellulaires :

$$\begin{aligned} J_i &= -M_i \nabla u_i, \\ J_e &= -M_e \nabla u_e, \end{aligned}$$

où M_i et M_e sont les conductivités respectives de chacun des milieux.

On suppose toujours qu'il n'y a pas de création de charge, mais qu'il peut y avoir une accumulation de charge en un point du domaine. Comme la membrane cellulaire est très fine, une accumulation de charge d'un côté de la membrane attire une quantité égale de charge opposée de l'autre côté de la membrane. Ainsi, en tout point, l'accumulation de charge totale est nulle, ce qui se traduit

$$\frac{\partial}{\partial t}(q_i + q_e) = 0,$$

où q_i et q_e représentent les charges intra (resp. extra) cellulaires.

Le courant net en un point du domaine est donné par la somme du taux d'accumulation de charge en point et du courant ionique sortant du domaine en ce point, autrement dit, on a

$$-\nabla \cdot J_i = \frac{\partial q_i}{\partial t} + \chi I_{ion}, \quad (2.2)$$

$$-\nabla \cdot J_e = \frac{\partial q_e}{\partial t} - \chi I_{ion}, \quad (2.3)$$

où χ représente l'aire de membrane cellulaire par unité de volume et I_{ion} le courant ionique à travers la paroi cellulaire. Le signe « $-$ » dans la seconde équation vient du fait que le sens positif du courant ionique est dirigé, par convention, de l'intérieur vers l'extérieur de la cellule.

On déduit des équations (2.2) et (2.3) que le courant total est conservé. En effet, on a

$$\begin{aligned} \nabla \cdot J_i + \nabla \cdot J_e &= -\frac{\partial q_i}{\partial t} - \chi I_{ion} - \frac{\partial q_e}{\partial t} + \chi I_{ion} \\ &= -\frac{\partial}{\partial t}(q_i + q_e) = 0. \end{aligned}$$

La quantité de charge pouvant être séparée par la membrane cellulaire dépend du potentiel électrique de celle-ci mais aussi de sa capacité C_m . En posant $v = u_i - u_e$ et $q = \frac{1}{2}(q_i - q_e)$, on

a la relation

$$v = \frac{q}{\chi C_m}.$$

Ainsi, on obtient

$$\begin{aligned} C_m \chi \frac{\partial v}{\partial t} &= \frac{\partial q}{\partial t} = \frac{1}{2} \frac{\partial(q_i - q_e)}{\partial t} \\ \Leftrightarrow \frac{\partial q_i}{\partial t} &= -\frac{\partial q_e}{\partial t} = C_m \chi \frac{\partial v}{\partial t} \\ \Leftrightarrow -\nabla \cdot J_i &= C_m \chi \frac{\partial v}{\partial t} + \chi I_{ion}. \end{aligned}$$

Finalement, en posant $u_i = v - u_e$, on obtient la formulation standard du modèle bidomaine :

$$\begin{cases} \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u_e) = C_m \chi \frac{\partial v}{\partial t} + \chi I_{ion}, \\ \nabla \cdot (M_i \nabla v) + \nabla \cdot ((M_i + M_e) \nabla u_e) = 0. \end{cases} \quad (2.4)$$

Remarque 2.1 (Géométrie de tissus cardiaques)

On peut noter que les tissus musculaires cardiaques présentent des qualités de conduction spécifiques. L'anisotropie vient du fait que le muscle est constitué de fibres qui se décomposent elles-mêmes en feuilles. En chaque point, on peut placer un repère orthonormal $\{a_l, a_t, a_n\}$ avec a_l dirigé selon une fibre, a_t et a_n perpendiculaires à $a_l - a_t$ dans le plan de la feuille et a_n dans le plan orthogonal à la feuille. La conductivité du milieu peut alors être écrite de manière tensorielle dans le repère local précédent comme

$$M^* = \begin{bmatrix} \sigma_l & 0 & 0 \\ 0 & \sigma_t & 0 \\ 0 & 0 & \sigma_n \end{bmatrix}.$$

Pour un champ électrique $E^* = (e_1, e_2, e_3)^T$ dans la base $\{a_l, a_t, a_n\}$, on a par la loi d'Ohm

$$J^* = M^* E^* = (\sigma_l e_1, \sigma_t e_2, \sigma_n e_3)^T = (j_1, j_2, j_3)^T$$

que l'on peut exprimer dans le système de coordonnée global

$$J = a_l j_1 + a_t j_2 + a_n j_3 = AJ^*.$$

Ainsi, on a

$$E^* = A^{-1} E,$$

avec $A^T = A^{-1}$, ce qui implique

$$J = AM^*A^T E.$$

Les tenseurs de conductivité sont donnés par

$$\begin{aligned} M_i &= AM_i^*A^T, \\ M_e &= AM_e^*A^T, \end{aligned}$$

où M_i^* et M_e^* sont les tenseurs des conductivité intra (resp. extra) cellulaire dans le repère local. Finalement, on constate que si l'on connaît en chaque point I_{ion} et $\{a_l, a_t, a_n\}$, on a une spécification complète des paramètres du modèle défini en (2.4).

Pour la résolution du système différentiel, il reste à définir les conditions aux bords pour les variables v et u_e . Dans ce but, on suppose que le cœur est entouré d'un milieu non conducteur, c'est-à-dire

$$n \cdot J_i = 0 \quad \text{et} \quad n \cdot J_e = 0,$$

sur le bord du domaine. Ainsi, pour un domaine Ω , on obtient le système aux dérivées partielles :

$$\begin{aligned} \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u_e) &= C_m \chi \frac{\partial v}{\partial t} + \chi I_{ion}, & \text{dans } \Omega, \\ \nabla \cdot (M_i \nabla v) + \nabla \cdot ((M_i + M_e) \nabla u_e) &= 0, & \text{dans } \Omega, \\ n \cdot (M_i \nabla v + M_i \nabla u_e) &= 0, & \text{sur } \partial\Omega, \\ n \cdot (M_e \nabla u_e) &= 0, & \text{sur } \partial\Omega. \end{aligned} \tag{2.5}$$

Remarque 2.2 (Existence d'une solution faible)

En imposant certaines conditions de régularité sur le domaine ainsi que sur les espaces fonctionnels utilisés, on peut montrer que, sous de « bonnes » hypothèses, le problème (2.5) possède une unique solution faible. Ce résultat demande l'introduction d'un formalisme adéquat ainsi que la démonstration de propriétés importantes. La démarche détaillée et complète qui permet l'établissement de ce résultat est présenté dans [CFS02].

2.2.3 Modèle monodomaine

Il est possible de simplifier le modèle bidomaine en supposant que les domaines intracellulaires et extracellulaires présentent le même type d'anisotropie, c'est-à-dire

$$M_e = \lambda M_i, \quad \lambda \in \mathbb{R}. \tag{2.6}$$

De cette manière, le système (2.5) peut se ramner à une équation scalaire. En effet, on a

$$\nabla \cdot (M_i \nabla u_e) = -\frac{1}{1+\lambda} \nabla \cdot (M_i \nabla v)$$

et par conséquent on se ramène à la résolution de :

$$\begin{aligned} \left(\frac{\lambda}{1+\lambda}\right) \nabla \cdot (M_i \nabla v) &= C_m \chi \frac{\partial v}{\partial t} + \chi I_{ion}, & \text{dans } \Omega, \\ n \cdot (M_i \nabla v) &= 0, & \text{sur } \partial\Omega. \end{aligned} \tag{2.7}$$

Remarque 2.3 (Modèle monodomaine anisotropique)

Il est possible de dériver le modèle monodomaine également sans l'hypothèse (2.6). On utilise pour cela l'expression du courant total $J_{tot} = -M_i \nabla u_i - M_e \nabla u_e$ et en posant $u_i = v + u_e$, on obtient

$$\nabla u_e = -M^{-1} M_i \nabla v - M^{-1} J_{tot},$$

avec $M = M_i + M_e$. De cette manière, on a

$$\nabla \cdot (M_i \nabla u_e) = \nabla \cdot (-M_i M^{-1} M_i \nabla v - M_i M^{-1} J_{tot}),$$

où J_{tot} dépend encore de u_e et u_i indépendemment. On cherche donc à approcher ce terme en fonction de v uniquement. On utilise pour cela la décomposition des tenseurs de conductivité dans le repère local $\{a_l, a_t, a_n\}$, voir Remarque 2.1. L'approximation consiste à négliger les projections de J_{tot} le long des directions perpendiculaires aux fibres, ce qui est autorisé par le fait que le courant est majoritairement colinéaire aux fibres. On utilise également que $\nabla \cdot J_{tot} = -I_{app}$. Les mêmes considérations sont employées pour déduire la condition de bord.

Le détail de cette méthode est rapporté dans [CFDE⁺06], section 2.3.

2.3 Modélisation du courant ionique

Pour résoudre les systèmes (2.5) et (2.7), il nous faut connaître l'expression de I_{ion} . C'est pourquoi, on s'intéresse maintenant à la description de ce courant. La spécification de celui-ci est essentiellement de deux types : phénoménologique ou physiologique. Les premiers modèles veulent reproduire le comportement macroscopique des cellules alors que les seconds se basent sur leur propriétés microscopiques extraites de données expérimentales.

Remarque 2.4 (Modèle cellulaire)

Les modèles que l'on présente ici se basent sur la modélisation d'une seule cellule. Dans cette situation, toutes les charges transportées par le courant ionique s'accumulent sur la membrane cellulaire. De cette manière, on peut voir celle-ci comme un condensateur et le potentiel électrique v de la membrane répond alors à l'équation

$$C_m \frac{dv}{dt} = -I_{ion} + I_{app},$$

où I_{app} est un stimulus appliqué à la cellule dans le but d'induire un potentiel d'action.

2.3.1 Modèles phénoménologiques

On commence par décrire deux modèles phénoménologiques. Ces modèles ont comme base la description suivante du courant ionique :

$$I_{ion} = f(v) = A^2(v - v_r)(v - v_t)(v - v_p), \quad (2.8)$$

où v_r est le potentiel au repos, v_t le potentiel de seuil et v_p le potentiel maximal. Ce modèle très simple permet de rendre compte de la plupart des phénomènes macroscopiques, mais bien sûr pas de tous. Notamment, il ne permet pas de décrire la phase de repolarisation des cellules. Dans ce but, des modèles plus complexes ont été développés.

FitzHugh-Nagumo (FHN) Pour obtenir une description qualitative correcte de cette phase de repolarisation, le modèle simpliste (2.8) est amélioré par l'ajout d'une variable de récupération w . Le modèle de FitzHugh-Nagumo [Fit61] [NAY62] est ainsi donné par le système différentiel

$$\begin{aligned} \frac{dv}{dt} &= c_1 v(v - a)(1 - v) - c_2 w + i_{app}, \\ \frac{dw}{dt} &= b(v - c_3 w), \end{aligned} \quad (2.9)$$

où a, b, c_1, c_2, c_3 sont des paramètres donnés, voir Table 1. Le stimulus I_{app} est appliqué ici entre $t = 50$ et $t = 60$ ms et son amplitude vaut 0.05. Le profil du potentiel électrique v et de la variable de lien w est illustré à la Figure 1. Maintenant que l'on connaît le courant ionique, on

| a | b | c_1 | c_2 | c_3 |
|------|-------|-------|-------|-------|
| 0.13 | 0.013 | 0.26 | 0.1 | 1.0 |

TABLE 1 – Paramètres pour les modèles de FitzHugh-Nagumo et Rogers-McCulloch.

peut écrire le modèle complet dans le cas monodomaine ou bidomaine. Dans le premier cas, on obtient

$$\begin{aligned} \frac{\lambda}{1+\lambda} M_i \Delta v \frac{\partial v}{\partial t} &= c_1 v(v - a)(1 - v) - c_2 w + i_{app} && \text{dans } \Omega, \\ \frac{\partial w}{\partial t} &= b(v - c_3 w) && \text{dans } \Omega, \\ M_i \frac{\partial v}{\partial n} &= 0 && \text{sur } \partial\Omega. \end{aligned} \quad (2.10)$$

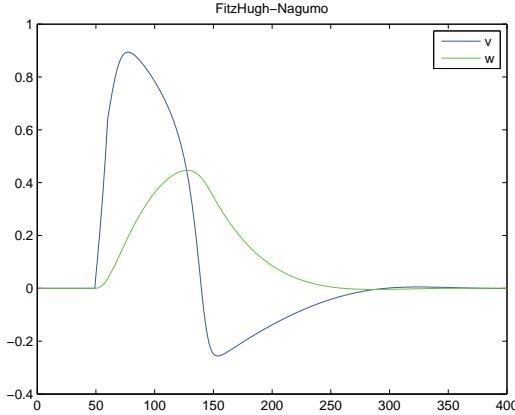


FIGURE 1 – Potentiel d’action issu du modèle de FitzHugh-Nagumo (2.9).

On dérive de la même manière la formulation du modèle dans le cas bidomaine.

Ce modèle est appréciable car il est assez simple et computationnellement facile à résoudre. Néanmoins, il présente des problèmes dans la description physiologique des cellules. S’il est très bon pour décrire le comportement dans les cellules neuronales, ce modèle ne l’est pas pour les cellules cardiaques. En effet, on observe une hyperpolarisation dans la phase de repolarisation et, de plus, le potentiel prend des valeurs inférieures à celle au repos ce qui n’est physiquement pas possible. Le modèle suivant permet de corriger cela.

Rogers-McCulloch (RM) La modification apportée par Rogers et McCulloch [RM94] dans leur nouveau modèle ne concerne que le dernier terme de la première équation du système (2.9) et permet d’éviter les abérrations physiologiques du modèle précédent :

$$\begin{aligned} \frac{dv}{dt} &= c_1 v(v - a)(1 - v) - c_2 v w + i_{app}, \\ \frac{dw}{dt} &= b(v - c_3 w), \end{aligned} \quad (2.11)$$

où a, b, c_1, c_2, c_3 sont des paramètres donnés, voir Table 1. Le potentiel obtenu cette fois-ci est qualitativement correct d’un point de vue physiologique, mais ses valeurs sont dans une échelle incorrecte, voir Figure 2. Pour rétablir des valeurs cohérentes, il nous faut transformer les variables v et w . Dans ce but, on exprime l’amplitude du potentiel d’action comme $v_{amp} = v_p - v_r$ et on pose le changement de variable :

$$\begin{aligned} V &= v_{amp}v + v_r, \\ W &= v_{amp}w, \\ I_{app} &= v_{amp}i_{app}, \\ v_t &= v_r + av_{amp}. \end{aligned}$$

Après le changement de variable, le système devient

$$\begin{aligned} \frac{dV}{dt} &= \frac{c_1}{v_{amp}^2}(V - v_r)(V - v_t)(v_p - V) - \frac{c_2}{v_{amp}}(V - v_r)W + I_{app}, \\ \frac{dW}{dt} &= b(V - v_r - c_3 W). \end{aligned} \quad (2.12)$$

Les valeurs de v_r , v_t et v_p sont données dans la Table 2. Ce modèle permet de décrire de manière correcte le potentiel d’action électrique des cellules cardiaques du point de vue de son amplitude, de sa durée et de sa vitesse d’élévation. Ceci est illustré sur l’image de droite de la Figure 2.

| v_r | v_t | v_p |
|-------|--------|-------|
| -85 | -68.75 | 40 |

TABLE 2 – Paramètres de reparamétrage du modèle de Rogers-McCulloch.

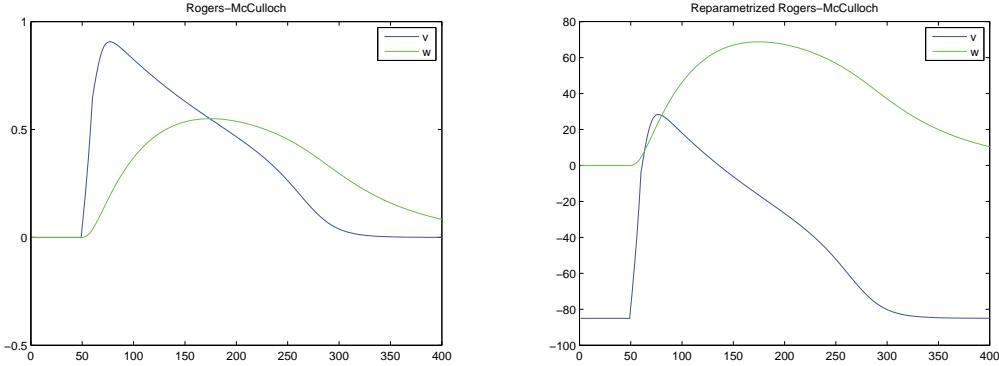


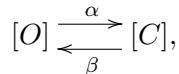
FIGURE 2 – Potentiels d’action obtenus avec le modèle de Rogers-McCulloch simple (2.11) à gauche et reparamétrisé (2.12) à droite.

2.3.2 Modèles physiologiques

Les deux modèles que l’on présente ensuite sont construits à partir du comportement physiologique des cellules cardiaques et en particulier des échanges ioniques. Le but du développement des modèles suivants est de comprendre l’influence des phénomènes au niveau cellulaire ou subcellulaire sur le comportement des tissu ou des organes. Les modèles suivants sont développés en ajustant des modèles plus ou moins complexes sur des jeux de données expérimentales. Avant de les présenter, il convient de s’intéresser au fonctionnement de la membrane cellulaire.

Echanges ioniques La membrane cellulaire est une paroi qui sépare l’espace extracellulaire du myoplasme, le liquide intracellulaire. Dans le but de maintenir la bonne concentration de chaque type de ion, elle permet des échanges ioniques. Elle présente des canaux qui ne laissent passer que certains types de ions. Ceux-ci peuvent être représentés comme des pompes ou des échangeurs. Les échanges ioniques permettent de générer et de maintenir une différence de potentiel le long de la membrane. L’ouverture et la fermeture de ces canaux dépend des concentrations ioniques ainsi que des champs électriques en présence. Ils s’ouvrent et se ferment en réponse à des stimuli électriques. Un tel canal est composé de plusieurs sous-unités qui peuvent alors chacunes être ouvertes ou fermées.

On note $[O]$ (resp. $[C]$) les concentrations ioniques présentes dans les canaux ouverts (resp. fermés). Le passage de l’état ouvert à l’état fermé peut être schématisé



où α est le degré d’ouverture du canal et β le degré de fermeture. Par la loi d’action de masse, voir [KS98], le degré de changement de l’état ouvert à l’état fermé est proportionnel à la concentration ionique du canal à l’état ouvert. Cela fonctionne de manière symétrique pour l’état fermé. Ceci se traduit

$$\frac{d[O]}{dt} = \alpha(v)[C] - \beta(v)[O]. \quad (2.13)$$

En posant $g = [O]/([O] + [C])$ et en divisant l'équation (2.13) par la concentration ionique totale $[O] + [C]$, on obtient

$$\frac{dg}{dt} = \alpha(v)(1 - g) - \beta(v)g. \quad (2.14)$$

De cette manière, on peut voir g comme la probabilité que le canal considéré soit ouvert. Par conséquent, si l'on considère un canal composé de n sous-unités équivalentes, la probabilité O que le canal soit ouvert est donnée par

$$O = g^n.$$

Un canal peut également être constitués de sous-unités de différentes natures avec chacune une probabilité g_i d'ouverture et des coefficients $\alpha_{g_i}, \beta_{g_i}$ déterminés par (2.14).

Etant donné O , on peut définir le courant qui traverse la membrane cellulaire. Il est donné par le produit du courant maximal avec la proportion de canaux ouverts. En posant G_{max} la conductance maximale, c.-à-d. la conductance lorsque tous les canaux sont ouverts, le courant est donné par

$$I = G_{max}O(v - v_{eq}), \quad (2.15)$$

où v_{eq} est le potentiel d'équilibre spécifique à chaque ion. Pour plus de détails sur les aspects physiologiques, on peut se référer à [KS98].

Les deux modèles que l'on présente ensuite sont basés sur l'étude de courants ioniques précis et utilisent pour ce faire leur expression de la forme (2.15).

Luo-Rudy, première génération (LR) Le modèle de Luo-Rudy [LR91] est un modèle pour le comportement des cellules ventriculaires. Il intègre la description de six courants ioniques avec sept canaux différents. En plus, des courants ioniques, ce modèle prend également en considération l'évolution de la concentration de calcium à l'intérieur de la cellule. Les quantités ioniques qui interviennent dans ce modèle sont résumées à la Figure 3.

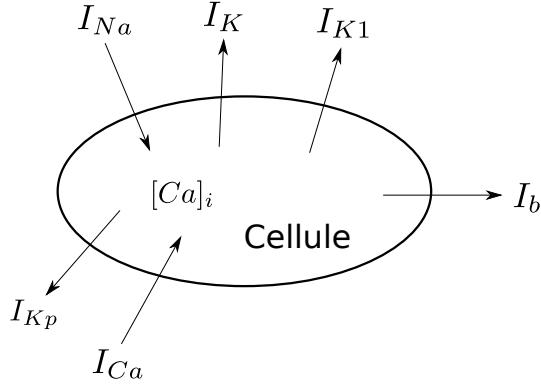


FIGURE 3 – Représentation schématique des interactions ioniques dans le cadre du modèle de Luo-Rudy pour les cellules ventriculaires.

Le modèle est décrit par le système de neuf équations différentielles :

$$\begin{aligned} -C_m \frac{dv}{dt} &= I_{ion} + I_{app}, \\ \frac{d[Ca]_i}{dt} &= -0.0001I_{Ca} + 0.07(0.0001 - [Ca]_i), \\ \frac{dg}{dt} &= \alpha_g(1 - g) - \beta_g g, \end{aligned}$$

avec $g = m, h, j, d, f, X, X_i$ les variables qui contrôlent les courants ioniques et

$$I_{ion} = I_{Na} + I_K + I_{K1} + I_{Kp} + I_{Ca} + I_b.$$

On peut trouver les paramètres de ce modèle dans [LR91]. Ce modèle se base sur le modèle de Beeler et Reuter qui est le premier à décrire le comportement des cellules ventriculaires, voir [BR77].

Luo-Rudy, deuxième génération (LR2) Le modèle précédent est amélioré pour inclure une description beaucoup plus précise des courants ioniques en présence. Ce modèle inclut douze courants ioniques différents. Il prend également en considération les concentrations ioniques intracellulaires de deux ions supplémentaires – le sodium et le potassium. Ces concentrations ont en effet une influence sur le potentiel électrique de la membrane. De plus, ce modèle décrit également certains flux intracellulaires. On ne donne pas ici les équations de ce modèle, pour plus de détails on peut se référer à [SLC⁺06].

Modèle Minimal (MM) Le but de ce modèle est de décrire au mieux les propriétés électrophysiologiques des ventricules tout en satisfaisant les deux conditions suivantes : premièrement, le modèle doit reproduire au mieux les données expérimentales extraites de différents tissus cardiaques humains et deuxièmement, le modèle doit être computationnellement efficace pour permettre la réalisation de simulations à grande échelle.

Le modèle minimal permet de reproduire le potentiel de la membrane au repos, le seuil d'excitation, la morphologie du potentiel d'action et la durée du potentiel d'action et est construit en ajustant un modèle sur des données expérimentales pour chacune de ces variables. Le modèle que l'on obtient utilise 4 variables – le potentiel membranaire u et 3 variables de « gating » v, w, s – et correspond à la résolution du système différentiel :

$$\begin{aligned}\frac{\partial u}{\partial t} &= \nabla \cdot (M \nabla u) - (I_{fi} + I_{so} + I_{si}), \\ \frac{\partial v}{\partial t} &= (1 - H(u - \theta_v))(v_\infty - v)/\tau_v^- - H(u - \theta_v)v/\tau_v^+, \\ \frac{\partial w}{\partial t} &= (1 - H(u - \theta_w))(w_\infty - w)/\tau_w^- - H(u - \theta_w)v/\tau_w^+, \\ \frac{\partial s}{\partial t} &= ((1 + \tanh(\kappa_s(u - u_s)))/2 - s)/\tau_s,\end{aligned}$$

où $H(x)$ est la fonction standardisée de Heaviside. Les expressions des courants ioniques, les fonctions qui ne dépendent pas du temps et les paramètres sont donnés dans [BOCF08]. On obtient le potentiel de la Figure 4. Ce modèle permet également de traiter de domaines hétérogènes en utilisant des paramètres différents pour les différentes couches du tissu cardiaque : endocardium, midcardium et epicardium, voir [BOCF08]. On trouve la dérivation détaillée et l'implémentation du modèle dans [BOCF08].

Courtemanche-Ramirez-Nattel (CRN) Si beaucoup de modèles existent pour la description du potentiel électrique dans les ventricules, il n'en est pas de même pour la description du potentiel dans les oreillettes. De plus, leur comportement est quelque peu différent. En effet, l'épaisseur des parois est nettement moins significative dans les oreillettes et la vitesse de conduction est largement supérieure dans les ventricules. Il existe aussi des différences importantes entre les espèces. En ce sens, le modèle que l'on présente ici se base sur des données expérimentales mesurées directement sur des cellules d'oreillettes humaines. Il a été développé par Courtemanche, Ramirez et Nattel en 1998, [CRN98].

Ce modèle est basé sur le modèle (LR2) et le courant ionique total est alors donné par

$$I_{ion} = I_{Na} + I_K + I_{Ca} + I_b + I_p, \quad (2.16)$$

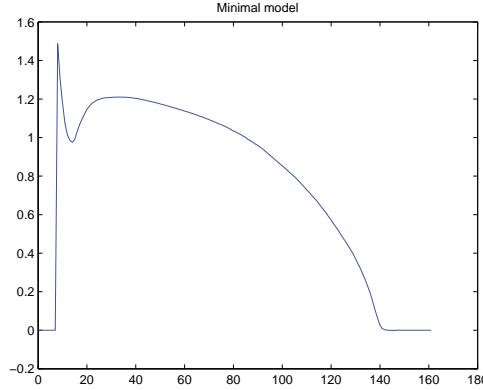


FIGURE 4 – Potentiel d’action issu du modèle minimal pour les ventricules.

Cette expression permet de rendre un nombre important d’aspects liés à la génération du potentiel et intègre les comportements associés aux ions du calcium, du sodium et du potassium.

Dans l’expression (2.16), le terme I_{Na} est le courant de dépolarisation rapide associé au ion Na^+ et la quantité I_K est le courant rectificateur total du ion K^+ . Ce courant ionique est composé de cinq courants est s’écrit

$$I_K = I_{K1} + I_{t0} + I_{Kur} + I_{Kr} + I_{Ks}.$$

Le courant I_{K1} décrit le courant rectificateur de K^+ entrant et joue un rôle important dans la description de la phase de repolarisation tardive du potentiel de l’oreillette ainsi que dans la détermination du potentiel au repos de la membrane. Le courant transitoire sortant de K^+ est donné par le terme I_{t0} . Les trois courants I_{Kur} , I_{Kr} et I_{Ks} représentent respectivement les courants rectificateurs ultra rapide, rapide et lent. Le terme I_{Ca} est le courant de type L du ion Ca^{2+} et $I_b = I_{b,Na} + I_{b,Ca}$ est le courant de fond associé aux ions Na^+ et Ca^{2+} . Enfin, I_p est le résultat des pompes et échangeurs et est construit de manière à remettre les concentrations ioniques à l’équilibre au repos. Il se décompose

$$I_p = I_{NaCa} + I_{NaK} + I_{p,Ca}.$$

Les différents courants ioniques sont décrits de manière plus détaillée dans [CRN98].

Le modèle de cellule utilisé ici comporte trois compartiments intracellulaires identiques à ceux utilisés dans le modèle (LR2). On a donc à nouveau une description des flux intracellulaires. De plus, ce modèle tient également en considérations les concentrations intracellulaires $[\text{Na}^+]_i$, $[\text{K}^+]_i$ et $[\text{Ca}^{2+}]_i$ ainsi que des concentration de calcium dans les différents compartiments intracellulaires.

L’équation de Nerst ([KS98]) nous permet de décrire le potentiel au repos pour un ion arbitraire X en fonction de sa conductance maximale g_X

$$E_X = \frac{RT}{zF} \log \frac{[X]_e}{[X]_i},$$

où R est la constante des gaz, T la température, F la constante de Faraday, $z = 1$ pour les ions Na^+ et K^+ , $z = 2$ pour le ion Ca^{2+} et $[X]_i$ (resp. $[X]_e$) sont les concentrations intra (resp. extra) cellulaires de ion X .

Toutes les concentrations ioniques en présence dépendent de la tension ainsi que, pour certaines, de l’ouverture et de la fermeture des canaux qui permettent le passage des ions à travers

la membrane cellulaire. Le comportement de ceux-ci est régit par des variables dite de « gating » qui répondent à l'équation différentielle ordinaire

$$\frac{dy}{dt} = -\frac{y^\infty - y}{\tau_y}, \quad (2.17)$$

où $y^\infty = \alpha_y(v)\tau_y(v)$ est la valeur de la variable au repos et $\tau_y(v) = \frac{1}{\alpha_y(v)+\beta_y(v)}$. Les variables $\alpha_y(v)$ et $\beta_y(v)$ sont spécifiques à chaque ion et sont décrites dans [CRN98].

L'évolution de la concentration intracellulaire de chacun des différents ions répond à des équations différentielles ordinaires. Pour leur formulation exacte, on peut se référer à [CRN98].

Globalement, le modèle se compose de 5 variables de concentration et 15 variables de « gating ». Celles-ci sont résumées avec leur courant ionique associé dans le Table 3. Les Figures 5, 6 et 7 représentent l'évolution des concentrations ioniques ainsi que des variables de « gating » lors d'une stimulation à 100 mV après 20 ms. On utilise $\Delta t = 0.005$ ms.

| Courant | I_{Na} | I_{t0} | I_{Kur} | I_{Kr} | I_{Ks} | $I_{Ca,L}$ | I_{rel} |
|-------------------------|-----------|----------|-----------|----------|----------|----------------|--------------|
| Variables de « gating » | m, h, j | aa, ai | ua, ui | xr | xs | d, f, f_{Ca} | pu, pv, pw |

TABLE 3 – Courants ioniques et variables de « gating » associées

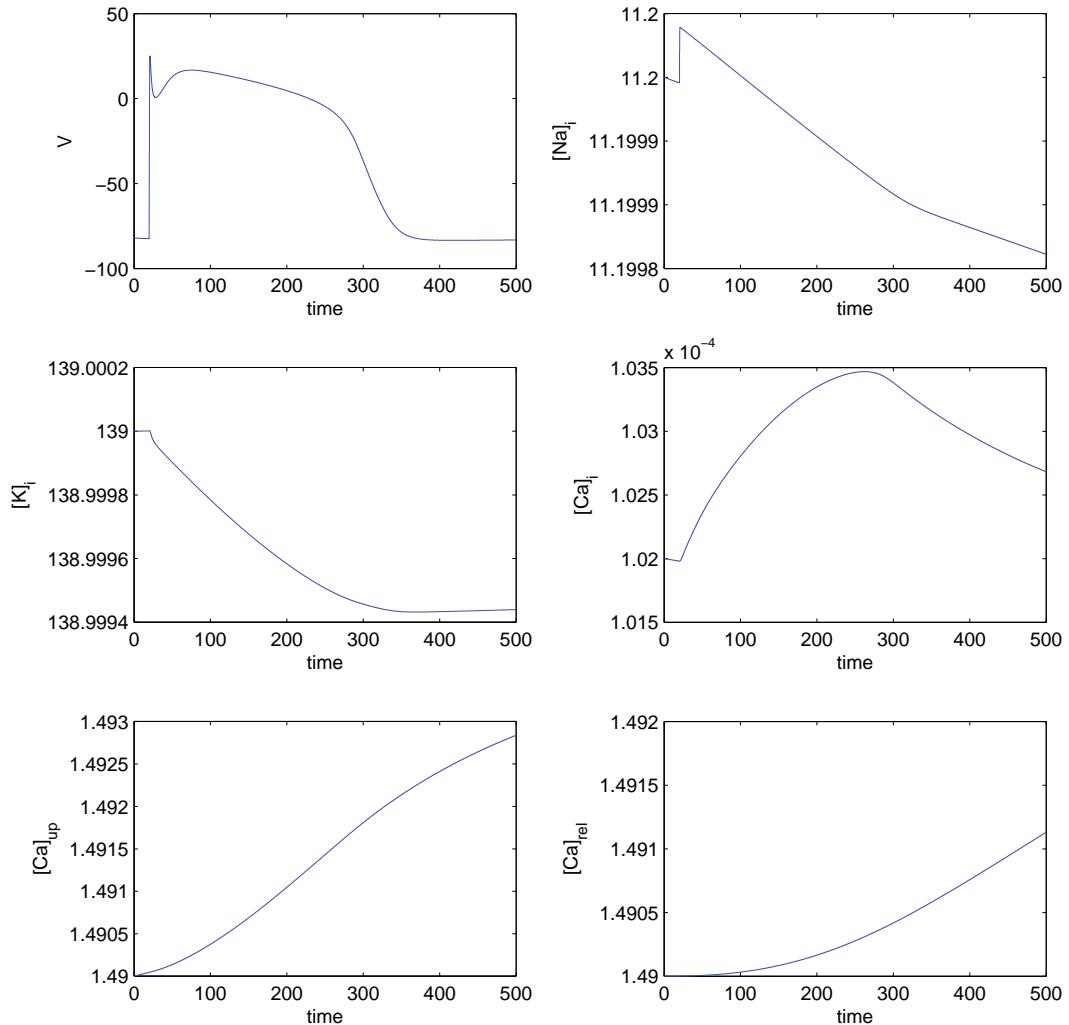


FIGURE 5 – Evolution du potentiel de la membrane et des concentrations ioniques pour le modèle CRN.

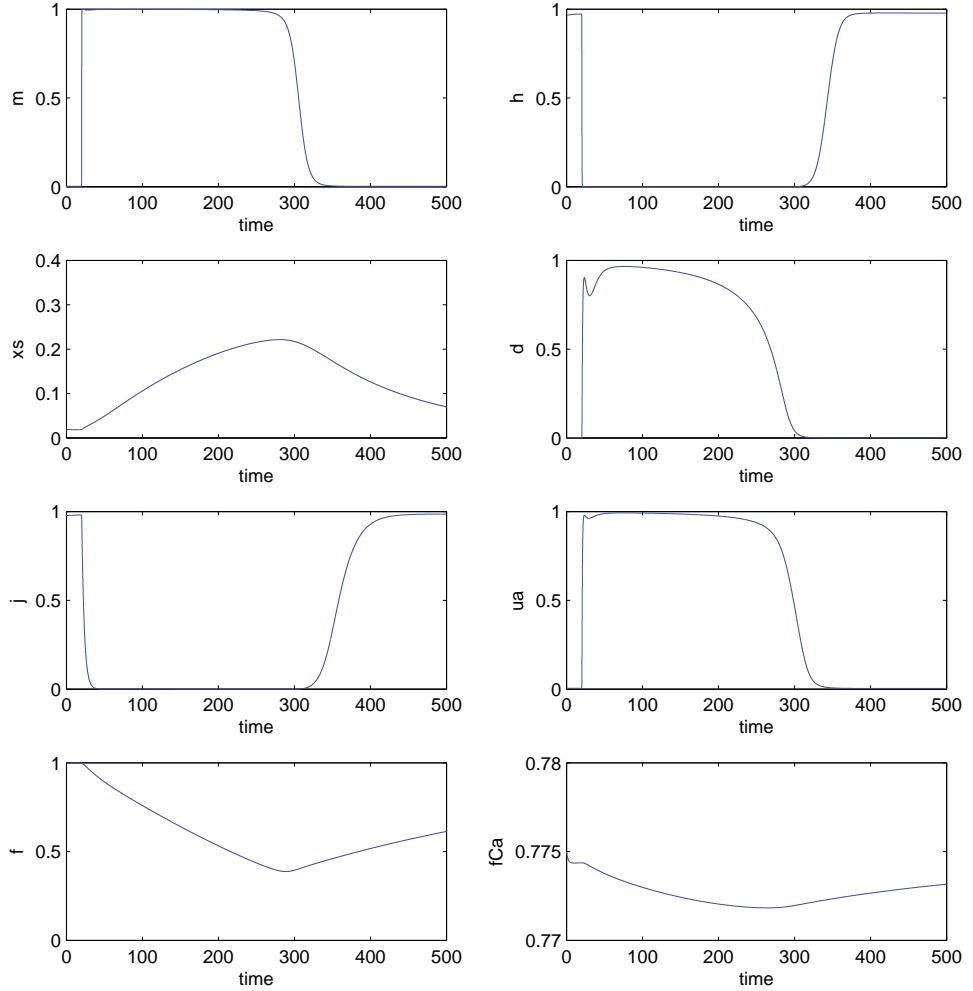


FIGURE 6 – Evolution des variables de « gating » pour le modèle CRN.

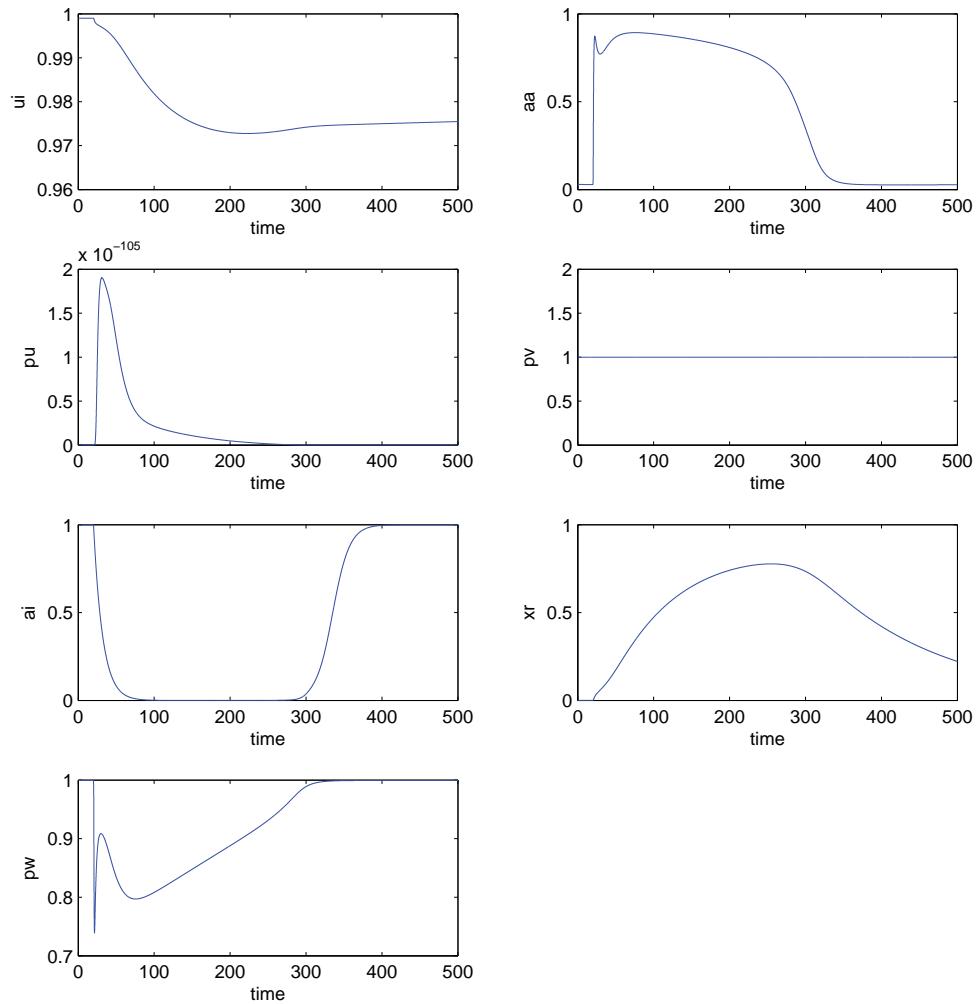


FIGURE 7 – Evolution des variables de « gating » pour le modèle CRN.

3 Discrétisation du modèle

On présente dans cette section la formulation variationnelle des équations bidomaines et monodomaines ainsi que leurs approximations de dimensions finies pour lesquelles on utilise l'espace de Sobolev H^1 sur \mathbb{R} . On trouve la définition de ce type d'espaces dans [Qua09].

Dans un second temps, on donne la semi-discrétisation en espace, puis la discrétisation complète des équations que l'on étudie. On se focalise ici sur le modèle ionique CRN qui est au centre de ce travail.

3.1 Formulation variationnelle

La formulation variationnelle du modèle bidomaine est donnée par : pour $v_0, w_0, c_0 \in L^2(\Omega)$, $I_{app} \in L^2(\Omega, (0, T))$, trouver $u_e(t), v(t) \in H^1(\Omega)$, tels que pour tout $t \in (0, T)$

$$\begin{cases} a_i(v(t), \phi) + a_i(u_e(t), \phi) + C_m \chi \left(\frac{\partial v(t)}{\partial t}, \phi \right) + \chi(I_{ion}(v(t), w(t)), c(t)) - (I_{app}, \phi) = 0 & \forall \phi \in H^1(\Omega) \\ a_i(v(t), \psi) + a_{i,e}(u_e(t), \psi) = 0 & \forall \psi \in H^1(\Omega) \\ v(t) = u_i(t) - u_e(t), \end{cases} \quad (3.1)$$

couplée avec les équations différentielles ordinaires qui régissent les concentrations ainsi que les variables de « gating ». On peut les écrire de manière générique comme

$$\begin{aligned} \frac{\partial c}{\partial t} &= S(v, w, c), \\ \frac{\partial w}{\partial t} &= R(v, w), \end{aligned}$$

avec les conditions initiales $w(0) = w_0$ et $c(0) = c_0$. La condition initiale pour le potentiel trans-membrane est donnée par $v(0) = v_0$. Dans (3.1), (\cdot, \cdot) dénote le produit scalaire dans $L^2(\Omega)$

$$(\eta, \xi) = \int_{\Omega} \eta \xi d\Omega, \quad \forall \eta, \xi \in L^2(\Omega)$$

et les formes variationnelles $a_i(\cdot, \cdot)$, $a_{i,e}(\cdot, \cdot)$ sont données par

$$a_x(\eta, \xi) = \int_{\Omega} M_x \nabla \eta \cdot \nabla \xi d\Omega, \quad \forall \eta, \xi \in H^1(\Omega),$$

avec $M_{i,e} = M_i + M_e$. Enfin, la formulation faible du problème monodomaine devient : pour $v_0, w_0, c_0 \in L^2(\Omega)$, $I_{app} \in L^2(\Omega, (0, T))$, trouver $v(t) \in H^1(\Omega)$, tel que pour tout $t \in (0, T)$

$$a(v(t), \phi) + C_m \chi \left(\frac{\partial v(t)}{\partial t}, \phi \right) + \chi(I_{ion}(v(t), w(t)), c(t)) - (I_{app}, \phi) = 0, \quad \forall \phi \in H^1(\Omega), \quad (3.2)$$

où la forme variationnelle $a(\cdot, \cdot)$ est cette fois-ci donnée par

$$a(\eta, \xi) = \int_{\Omega} \left(\frac{\lambda}{1 + \lambda} \right) M_i \nabla \eta \cdot \nabla \xi d\Omega, \quad \forall \eta, \xi \in H^1(\Omega).$$

Comme nous l'avons déjà mentionné à la Remarque 2.2, il est possible de montrer certains résultats concernant l'existence et l'unicité de solutions faibles pour les problèmes (3.1) et (3.2) ainsi que concernant la régularité des solutions, voir [CFS02].

3.2 Semi-discrétisation en espace

Soit \mathcal{T}_h une triangulation conforme du domaine Ω telle que $\Omega = \cup_{j=1}^N K_j$, avec $K_j \in \mathcal{T}_h$ pour $1 \leq j \leq N$. On suit ici la méthode de Galerkin [Qua09] pour obtenir la semi-discrétisation en espace des problèmes variationnels (3.1) et (3.2). Pour ce faire, on utilise l'espace d'éléments finis de degré $1 - \mathbb{P}_1 - X_h$ de dimension finie N_h . On note $\{\varphi\}_{i=1}^{N_h}$ la base de cet espace. On construit les matrice de masse $M = (m_{kl})$ et de raideur $A_{i,e} = (a_{kl}^{i,e}), A_i = (a_{kl}^i), A = (a_{kl})$ dont les éléments sont définis par

$$\begin{aligned} m_{kl} &= \sum_{j=1}^N \int_{K_j} \varphi_k \varphi_l d\Omega, \\ a_{kl}^{i,e} &= \sum_{j=1}^N \int_{K_j} M_{i,e} \nabla \varphi_k \cdot \nabla \varphi_l d\Omega, \\ a_{kl}^i &= \sum_{j=1}^N \int_{K_j} M_i \nabla \varphi_k \cdot \nabla \varphi_l d\Omega, \\ a_{kl} &= \sum_{j=1}^N \int_{K_j} \left(\frac{\lambda}{1+\lambda} \right) M_i \nabla \varphi_k \cdot \nabla \varphi_l d\Omega. \end{aligned}$$

On pose encore

$$\begin{aligned} u_{i,h} &= \sum_{j=1}^{N_h} u_{i,j}(t) \varphi_j, \quad u_{e,h} = \sum_{j=1}^{N_h} u_{e,j}(t) \varphi_j, \quad v_h = \sum_{j=1}^{N_h} v(t) \varphi_j, \\ w_h &= \sum_{j=1}^{N_h} w(t) \varphi_j, \quad c_h = \sum_{j=1}^{N_h} c(t) \varphi_j, \end{aligned}$$

que l'on note

$$\begin{aligned} \mathbf{u}_{i,h} &= (u_{i,1}, \dots, u_{i,N_h})^T, \quad \mathbf{u}_{e,h} = (u_{e,1}, \dots, u_{e,N_h})^T, \quad \mathbf{v}_h = (v_1, \dots, v_{N_h})^T, \\ \mathbf{w}_h &= (w_1, \dots, w_{N_h})^T, \quad \mathbf{c}_h = (c_1, \dots, c_{N_h})^T. \end{aligned}$$

Ainsi, on peut écrire la formulation semi-discrétisée en espace sous la forme matricielle suivante :

$$C_m \chi \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \frac{\partial}{\partial t} \begin{bmatrix} \mathbf{v}_h \\ 0 \end{bmatrix} + \begin{bmatrix} A_i & A_i \\ A_i & A_{i,e} \end{bmatrix} \begin{bmatrix} \mathbf{v}_h \\ \mathbf{u}_{e,h} \end{bmatrix} + \chi \begin{bmatrix} MI_{ion}(\mathbf{v}_h, \mathbf{w}_h, \mathbf{c}_h) \\ 0 \end{bmatrix} = \begin{bmatrix} MI_{app} \\ 0 \end{bmatrix}. \quad (3.3)$$

De la même manière, on obtient la forme matricielle du problème monodomaine :

$$C_m \chi M \frac{\partial \mathbf{v}_h}{\partial t} + A \mathbf{v}_h + MI_{ion}(\mathbf{v}_h, \mathbf{w}_h, \mathbf{c}_h) = MI_{app}. \quad (3.4)$$

Les équations matricielles (3.3) et (3.4) sont couplées avec les équations de « gating » et de concentration

$$\frac{\partial \mathbf{w}_h}{\partial t} = R(\mathbf{v}_h, \mathbf{w}_h), \quad \frac{\partial \mathbf{v}_h}{\partial t} = S(\mathbf{v}_h, \mathbf{w}_h, \mathbf{c}_h)$$

3.3 Discrétisation totale

Dans le but d'obtenir une approximation discrète complète des problèmes bidomaine et monodomaine, on intègre en temps les systèmes (3.3) et (3.4) avec un schéma d'Euler semi-implicite : le terme linéaire de diffusion est traité implicitement mais le courant ionique I_{ion} est mis à jour explicitement. Les variables de concentration sont intégrée grâce au schéma d'Euler rétrograde

$$\frac{\mathbf{c}_h^{n+1} - \mathbf{c}_h^n}{\Delta t} = S(\mathbf{v}_h^{n+1}, \mathbf{w}_h^{n+1}, \mathbf{c}_h^{n+1})$$

et l'intégration des variables de « gating » est obtenue après linéarisation autour du potentiel à l'intégration précédente par

$$\mathbf{w}_h^{n+1} = \mathbf{w}_\infty(\mathbf{v}_h^n) + (\mathbf{w}_h^n - \mathbf{w}_\infty(\mathbf{v}_h^n)) \exp\left(-\frac{\Delta t}{\tau \mathbf{w}_h^n(\mathbf{v}_h^n)}\right),$$

où les notations sont les mêmes qu'en (2.17) et proviennent du formalisme de Hodgkin-Huxley [KS98]. Ainsi, connaissant le potentiel l'itération précédente, il est possible de résoudre les systèmes d'équations ordinaires concernant les concentrations et les variables de « gating ». Donc, connaissant le potentiel au pas de temps précédent, le problème bidomaine devient : trouver \mathbf{v}_h^{n+1} qui satisfait

$$\begin{cases} C_m \chi M \frac{\mathbf{v}_h^{n+1} - \mathbf{v}_h^n}{\Delta t} + A_i(\mathbf{v}_h^{n+1} + \mathbf{u}_{e,h}^{n+1}) + MI_{ion}(\mathbf{v}_h^n, \mathbf{w}_h^{n+1}, \mathbf{c}_h^{n+1}) = MI_{app}, \\ A_i \mathbf{v}_h^{n+1} + A_{i,e} \mathbf{u}_{e,h}^{n+1} = 0 \end{cases}$$

et le problème monodomaine : trouver \mathbf{v}_h^{n+1} qui satisfait

$$C_m \chi M \frac{\mathbf{v}_h^{n+1} - \mathbf{v}_h^n}{\Delta t} + A \mathbf{v}_h^{n+1} + MI_{ion}(\mathbf{v}_h^n, \mathbf{w}_h^{n+1}, \mathbf{c}_h^{n+1}) = MI_{app}.$$

Ce qui amène, dans le cas bidomaine, à résoudre le système linéaire

$$\begin{bmatrix} C_m \chi M + \Delta t A_i & \Delta t A_i \\ A_i & A_{i,e} \end{bmatrix} \begin{bmatrix} \mathbf{v}_h^{n+1} \\ \mathbf{u}_{e,h}^{n+1} \end{bmatrix} = -C_m \chi \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_h^n \\ \mathbf{u}_{e,h}^n \end{bmatrix} - \begin{bmatrix} \Delta t MI_{ion}(\mathbf{v}_h^n, \mathbf{w}_h^{n+1}, \mathbf{c}_h^{n+1}) \\ 0 \end{bmatrix} + \begin{bmatrix} \Delta t MI_{app} \\ 0 \end{bmatrix} \quad (3.5)$$

et pour le cas monodomaine

$$(C_m \chi M + \Delta t A) \mathbf{v}_h^{n+1} = C_m \chi \mathbf{v}_h^n - \Delta t A \mathbf{v}_h^n - \Delta t MI_{ion}(\mathbf{v}_h^n, \mathbf{w}_h^{n+1}, \mathbf{c}_h^{n+1}) + \Delta t MI_{app}. \quad (3.6)$$

Remarque 3.1

Il est possible d'utiliser l'algorithme du gradient préconditionné pour résoudre les systèmes (3.5) et (3.6) en utilisant comme condition initiale le potentiel à l'itération précédente. On trouve plus de détails sur cette méthode dans [QSS07].

4 Résultats numériques

Le but de cette section est de présenter les différents résultats numériques obtenus lors de simulations. Les simulations bidimensionnelles sont calculées avec FreeFEM++, [Hec09] et les tridimensionnelles avec LifeV, [Lif].

On utilise dans un premier temps le modèle cellulaire RM, puis CRN dont on cherche à valider le code LifeV. Pour ces deux modèles, on effectue des simulations uniquement dans le cas monodomaine.

Pour toutes les simulations effectuées dans la suite du travail, on utilise des éléments finis de type \mathbb{P}_1 pour tous les champs scalaires. De plus, on utilise le solveur GMRES avec les paramètres suivants :

- tolérance : 10^{-7} ;
- nombre maximal d’itérations : 100 ;
- dimension des espaces de Krylov : 200. Pour plus de détails sur cette méthode de résolution, on peut se référer à [QSS07]. On utilise le schéma d’Euler rétrograde pour l’intégration en temps.

On utilise dans tous les cas le tenseur de conductivité

$$\begin{bmatrix} 0.005 & 0 \\ 0 & 0.005 \end{bmatrix}$$

et on considère des milieux sans fibres.

Remarque 4.1 (LifeV)

Un des objectifs de ce projet est l’installation et l’utilisation de la librairie LifeV. Celle-ci permet l’implémentation de méthodes numériques pour la résolution de problèmes tridimensionnels aux éléments finis. Ainsi, le premier contact avec LifeV est sa compilation et son installation. Une fois cette étape franchie, on peut envisager la simulation de premiers problèmes simples.

4.1 Modèle RM

Le but de cette première simulation est de valider le modèle implémenté grâce à la librairie LifeV. Dans ce but, on effectue une comparaison des simulations avec LifeV et FreeFEM++ dans le cas d’un domaine carré.

On choisit un domaine carré $\Omega = [0, 5] \times [0, 5]$ avec 32 noeuds sur chaque bord pour la construction du domaine. LifeV ne permettant que des simulations 3D, on utilise un parallélépipède à base carrée avec également 32 noeuds sur chaque bord et d’une hauteur minime, correspondant à un 1 noeud. Le domaine est décrit par

$$\Omega = [0, 5] \times [0, 5] \times [0, 0.3125] \quad (4.1)$$

et possède 12'288 éléments, ce qui représente 3287 degrés de liberté. Il est présenté en Figure 8. On effectue une simulation sur un intervalle de temps de 600 ms avec une première stimulation en $(0, 0)$ au début de la simulation, puis une seconde en $t = 300$ ms au centre du domaine en $(2.5, 2.5)$. Les deux stimulations se font à 110 mV. Le code FreeFEM qui permet cela est rapporté en A.1.

On calcule ensuite sur ce domaine l’évolution du potentiel électrique grâce au modèle de Rogers-McCulloch monodomaine. Le but est de vérifier que les résultats que l’on obtient avec LifeV sont corrects. Pour ce faire, on effectue une simulation pour $0 \leq t \leq 600$ ms avec un potentiel initial $v_0 = -84$ mV et on compare les résultats obtenus avec les deux programmes. Le pas d’intégration est dans les deux cas $\Delta t = 0.5$ ms. On utilise le résultat obtenu avec FreeFEM

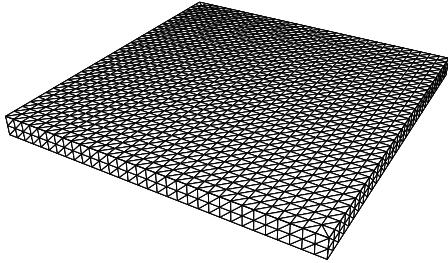


FIGURE 8 – Maillage du carré $\Omega = [0, 5] \times [0, 5]$ avec 32 noeuds sur chaque bord.

comme référence. La Figure 9 montre le potentiel obtenu dans chacune des deux situations. On remarque ainsi que l'on obtient comme voulu des résultats similaires. De cette manière on peut valider la simulation effectuées avec LifeV.

On peut aussi observer l'évolution du potentiel électrique en des points choisis du domaine, voir Figure 10. En observant ces graphes, on constate que l'évolution est relativement semblable entre les deux programmes. De plus, on constate que la forme de la courbe de potentiel est conforme à ce que l'on a établi de manière théorique, voir Figure 2.

Finalement, la résolution de ce problème avec LifeV se fait à la vitesse suivante : le système est résolu en une itération ce qui prend environ 0.004703 seconde en moyenne et une itération de temps complète est faite en environ 0.82 seconde.

4.2 Validation du code CRN

La validation du code implémenté dans LifeV pour le modèle CRN se fait en plusieurs étapes. On se base sur les résultats de [GG08] pour valider notre modèle.

Premièrement, on vérifie seulement le fonctionnement du modèle ionique sans inclure de terme de diffusion. Dans ce but, on résout

$$C_m \frac{dv}{dt} = -I_{ion} + I_{app},$$

pour I_{ion} donné par (2.16) et un stimulus de 100 mV appliqué après 20 ms. La résolution de ce premier problème est faite avec Matlab, voir B.1. Ceci permet d'obtenir les graphes des Figures 5 à 7. On remarque que les courbes de ces figures correspondent à celle du document de référence [GG08]. De cette manière, on peut supposer que celles-ci sont correctes. Cependant, afin d'obtenir ces résultats, il a fallu effectuer quelques corrections dans le code `HeartIonicSolver.hpp` de LifeV. En effet, ce code n'ayant pas été testé, il présente encore quelques erreurs. La partie corrigée du code est donnée en B.2.

Finalement, afin de prendre en considération le modèle CRN comme choix possible de modèle ionique, il faut ajouter l'extrait de code

```

1 else if ( dataFile("electric/physics/ion_model",1) == 5)
2 {
3     M_diffusivity = dataFile("electric/physics/D" , 5.7e-4);
4     M_longitudinalConductivity = dataFile("electric/physics/sigmal" , 1.2e-3);
5     M_transversalConductivity = dataFile("electric/physics/sigmat" , 2.56e-4);
6 }
```

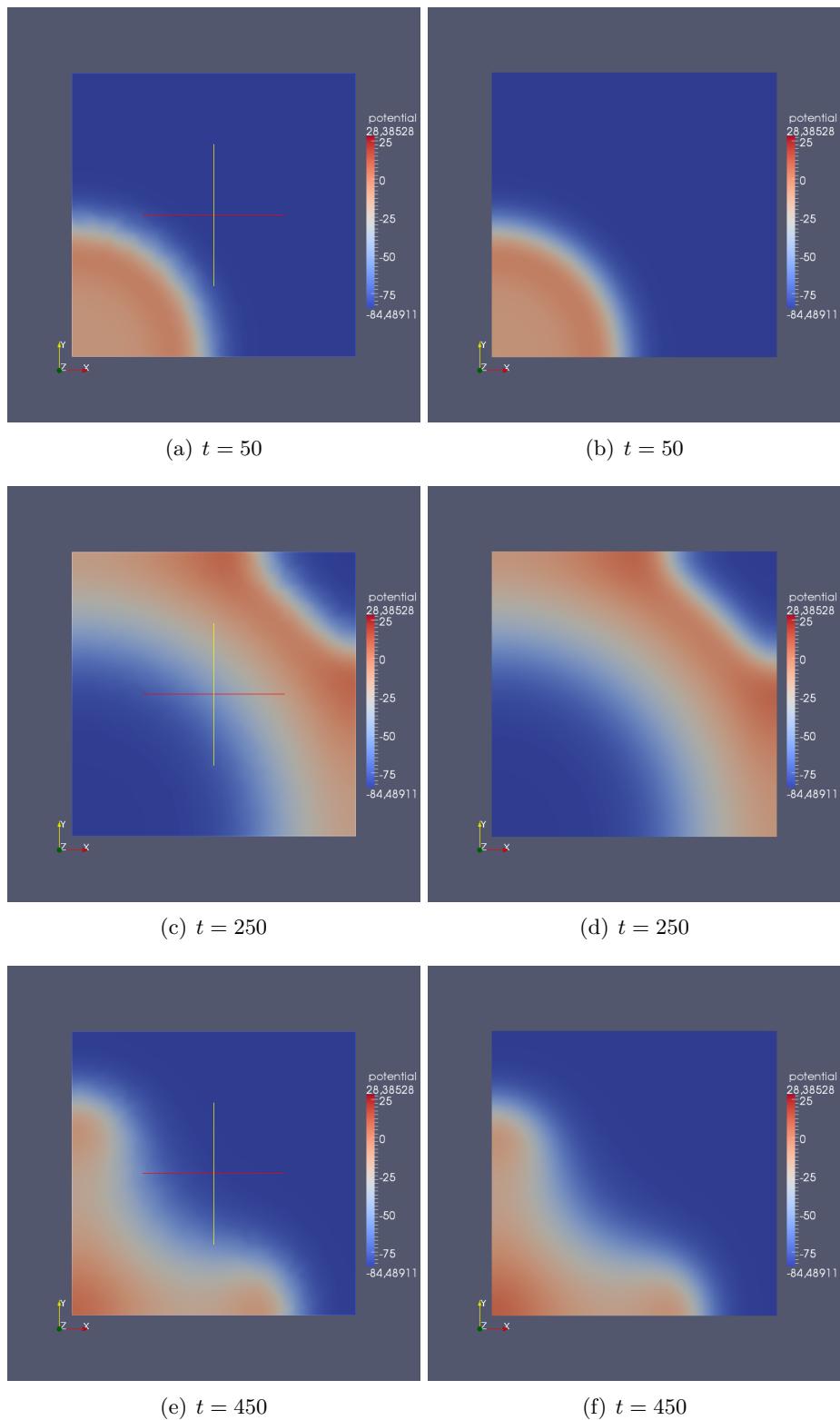


FIGURE 9 – Potentiel électrique calculé avec FreeFEM (en haut) et LifeV (en bas).

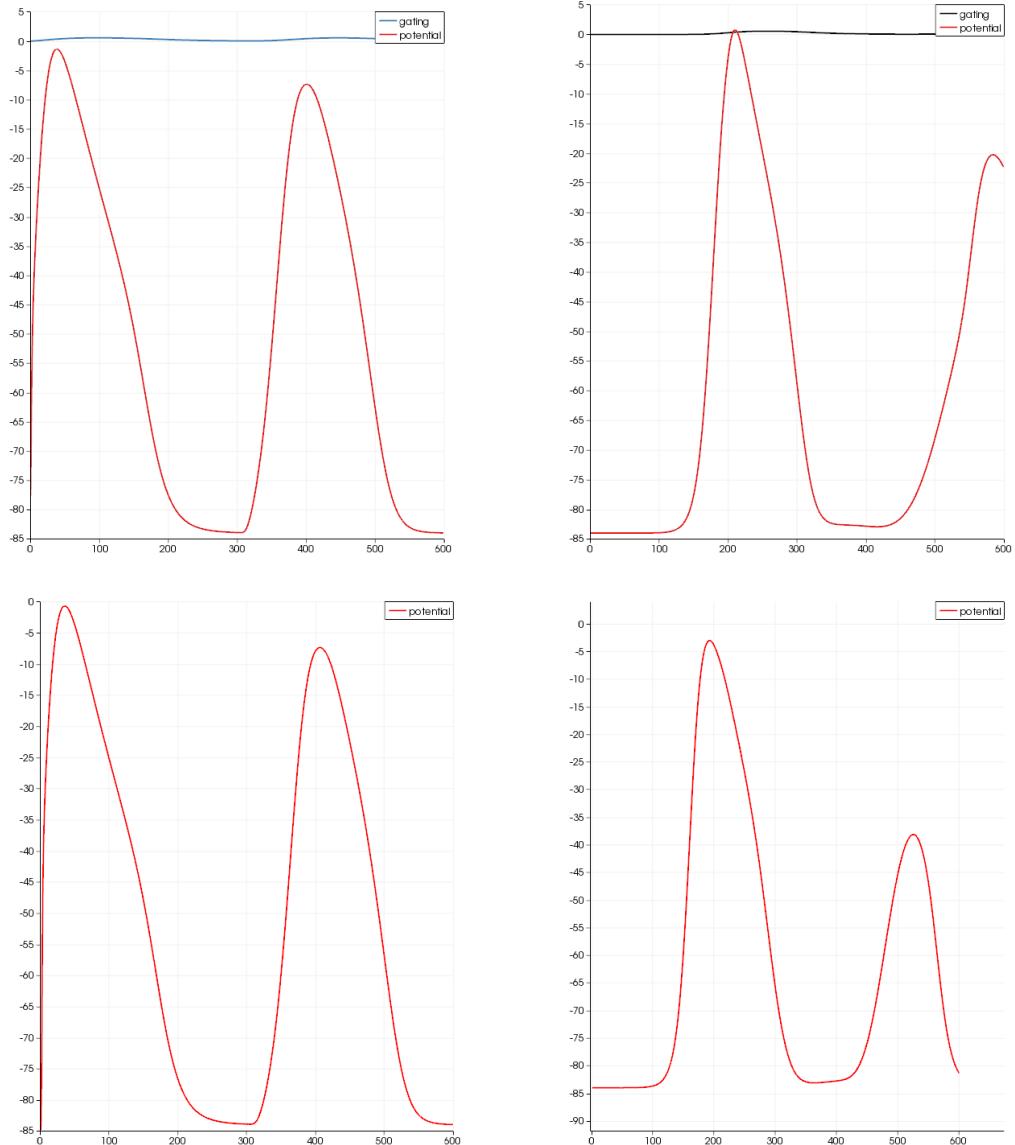


FIGURE 10 – Evolution du potentiel électrique en un point du domaine : avec FreeFEM (en haut) et LifeV (en bas) pour des point situés proches des coins gauche inférieur (à gauche) et supérieur (à droite).

dans le fichier `HeartMonodomainData.cpp` ainsi qu'effectuer les modifications nécessaires dans le fichier `data`.

Une fois ces quelques modifications effectuées, on peut tester le code avec le terme de diffusion. Pour ce faire, on effectue une simulation sur le domaine défini par (4.1) et de hauteur minime. Ceci nous permet de comparer les résultats que l'on obtient avec ceux de [GG08]. Il est vrai que le domaine employé dans cet article est un carré bidimensionnel, mais comme LifeV ne permet que la résolution de problèmes tridimensionnels, il nous faut ajouter une hauteur minime au carré de référence. On peut encore noter une autre différence avec les résultats de l'article : celui-ci emploie des fibres. Autrement dit, la vitesse de diffusion du potentiel électrique n'est pas la même selon toutes les directions de l'espace. Comme nous n'avons pas la description des fibres pour le maillage que l'on considère, on utilise comme facteur de diffusion D , la résultante des facteurs σ_l et σ_t de diffusion dans chacune des directions.

La Figure 11 montre l'évolution du potentiel électrique dans le domaine choisi. La simulation dure 500 ms avec un pas d'intégration de $\Delta t = 0.05$ ms et un stimulus de 100 mV est appliqué en $(0, 0)$ au temps $t = 20$ ms. On remarque que les résultats obtenus sont cohérents et correspondent à ceux de [GG08]. Finalement, ce test nous permet de vérifier que le code CRN de la librairie LifeV est correct et on peut ainsi l'utiliser pour la résolution de problèmes plus complexes.

4.3 Comparaison entre les modèles RM et CRN

Maintenant que l'on a vérifié l'implémentation des modèles RM et CRN, on peut s'intéresser à comparer leur comportement dans une situation identique. De cette manière, on effectue la même simulation que celle effectuée en 4.2 mais en utilisant le modèle ionique RM. On obtient les résultats résumés par la Figure 12.

On observe une différence évidente dans la diffusion du potentiel électrique selon le modèle ionique utilisé. On remarque qu'avec le modèle RM le pic de potentiel qui intervient lors de la stimulation est bien rendu mais le retour au potentiel au repos est beaucoup plus rapide. Ceci est attendu et est conforme au graphe obtenu à la Figure 2. On remarque aussi que la diffusion du potentiel est plus lente avec ce premier modèle que lorsque l'on utilise le modèle CRN. Dans ce second cas, le potentiel se diffuse de manière très rapide à travers le domaine et l'ensemble du domaine atteint le potentiel maximal au temps $t = 150$ ms environ. Le retour au potentiel au repos est plus lent et la décroissance est sensiblement simultanée pour l'ensemble des points du domaine contrairement à ce que l'on observe avec le modèle RM. En d'autres termes, le modèle CRN permet de mieux rendre la phase de plateau qui intervient après le pic de potentiel. On observe aussi cela à la Figure 5. Les temps de calcul sont similaires entre les deux modèles : une itération de temps complète se fait en une moyenne de 0.9 seconde.

4.4 Instigation d'ondes spirales

Maintenant que l'on sait nos codes valides sur un domaine simple, on peut les utiliser sur des domaines plus complexes. De cette manière, on regarde le comportement du code RM sur l'oreillette gauche.

Remarque 4.2 (Maillage)

Nous avons deux maillages à notre disposition pour effectuer les simulations sur la géométrie de l'oreillette gauche, voir Figure 13. Le premier est largement plus grossier que le second. En effet, le premier maillage est inclu dans le domaine $[-14.3755, 49.5128] \times [-150.568, -113.868] \times [1382.53, 1420.29]$ et possède 14'172 éléments alors que le second est de dimension similaire – inclus dans le domaine $[-71.8408, 54.8726] \times [-180.358, -101.318] \times [1329.58, 1420.11]$ – avec 947'632 éléments. En terme de degrés de liberté, cela en représente 4586 pour le premier et

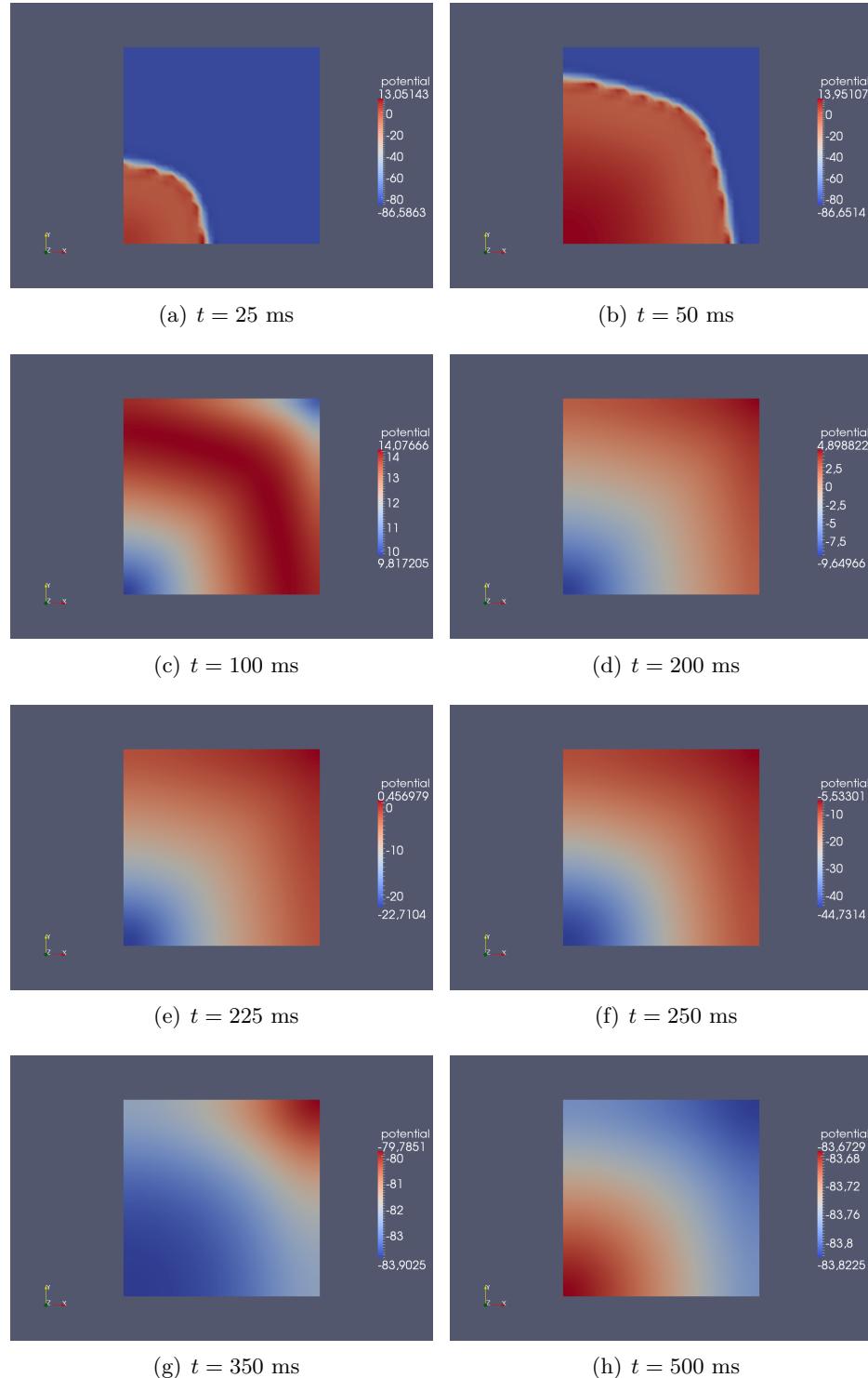


FIGURE 11 – Potentiel électrique calculé avec le modèle monodomaine CRN.

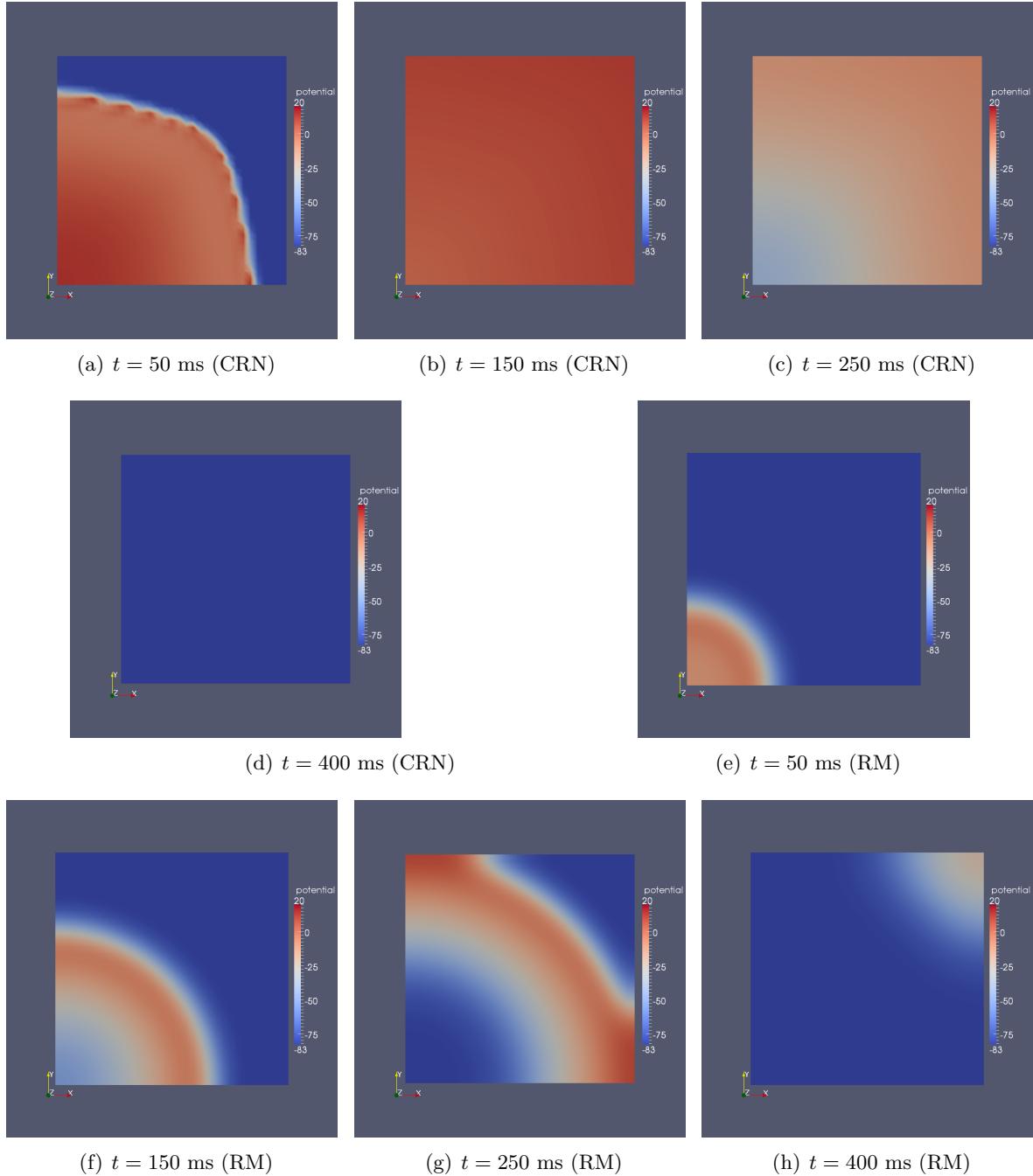


FIGURE 12 – Comparaison de la propagation d'une onde de potentiel pour les modèles RM et CRN.

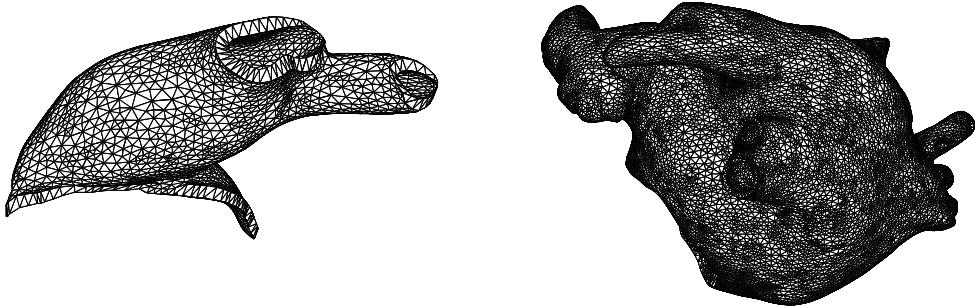


FIGURE 13 – Maillages de l’oreillette gauche grossier (à gauche) et fin (à droite).

171'022 pour le second. L’utilisation du premier maillage induit un temps de calcul de 0.9 seconde par itération de temps alors que le plus fin demande 190 secondes. Ainsi, pour des soucis de temps de calculs, on utilise uniquement le maillage le plus grossier, mais ces simulations peuvent faire l’objet d’une étude future.

On observe ici la propagation du potentiel lors de stimulations en plusieurs zones de l’oreillette. On a vu précédemment, lors de l’étude dans le domaine carré, qu’une stimulation ponctuelle se traduit par la génération d’une onde plane. Ce type d’onde est synonyme d’une activité cardiaque saine alors que la présence d’une onde spirale peut dénoter une tachycardie ou même aboutir à une entrée en fibrillation du tissu considéré. Dans la mesure où la compréhension des mécanismes de déclenchement de ce type de processus anormal peut aider à la prévention de celui-ci, on montre ici comment instaurer une onde spirale dans l’oreillette gauche. Pour ce faire, on applique successivement deux stimuli : le second se place à l’intérieur de la zone de propagation du premier. De cette manière, on parvient à générer l’onde spirale périodique présentée à la Figure 14.

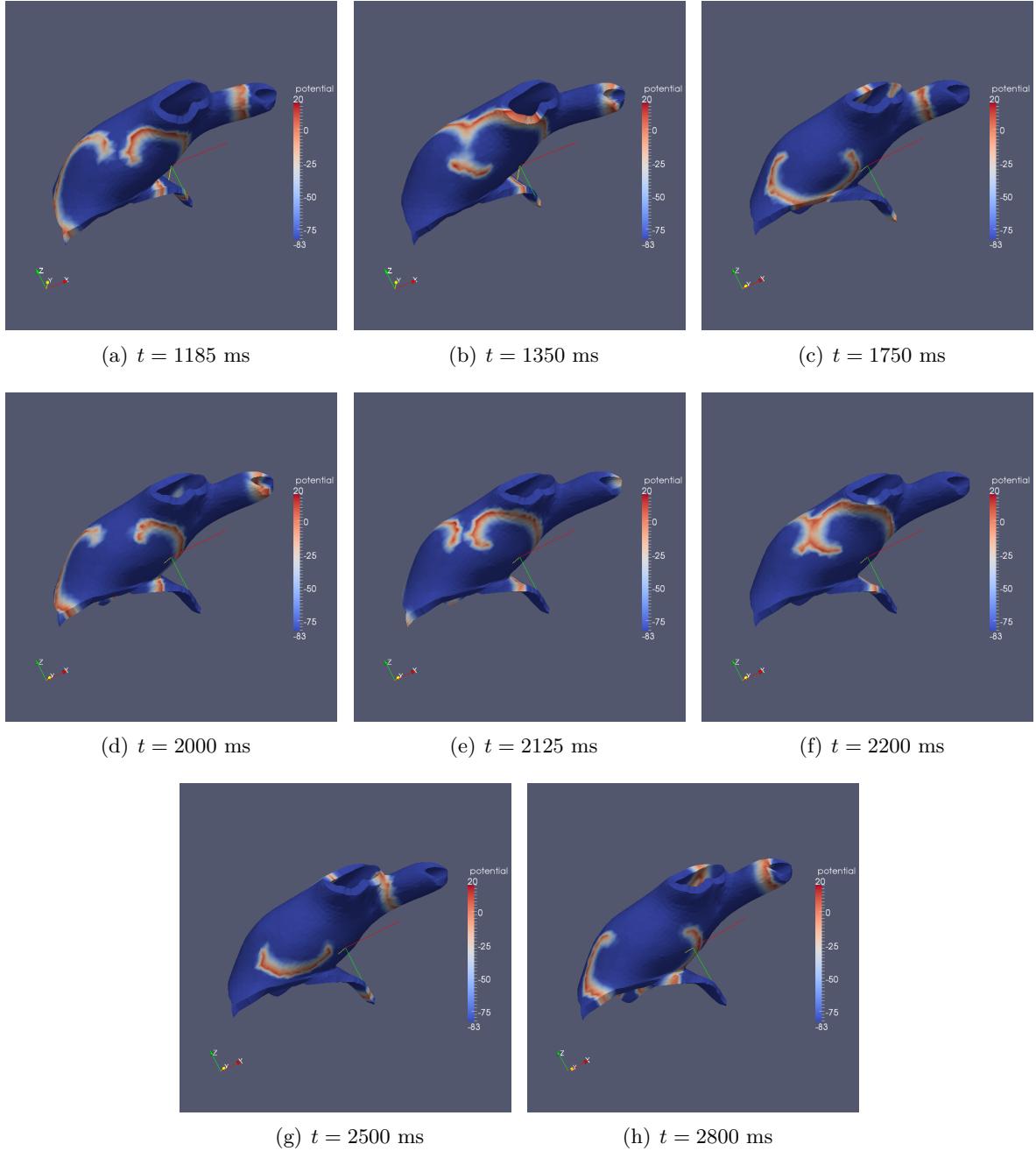


FIGURE 14 – Génération d’onde spirale « périodique » dans l’oreille gauche, modèle RM.

Deuxième partie

Contrôle optimal en électrophysiologie cardiaque

5 Problème inverse en électrocardiographie

On parle du problème inverse en opposition au problème direct : le problème direct calcule l'effet d'une cause définie, alors que le problème inverse cherche la cause d'un effet donné. Le premier est bien posé, ce qui n'est pas le cas du second.

Le problème inverse classique en électrocardiographie consiste à récupérer le potentiel électrique à la surface du cœur à partir de données à la surface du torse du patient. En effet, le dispositif classique de mesure de l'activité cardiaque d'un patient est l'électrocardiogramme. Celui-ci consiste en des électrodes disposées en différents endroits du torse et qui enregistrent l'activité électrique à cet endroit. On veut donc reconstruire le potentiel électrique cardiaque alors inaccessible à partir de ces données. Différentes méthodes de résolution de ce problème sont présentées dans [PCN⁺11] et [SLC⁺06].

La résolution de ce problème amène à des perspectives très intéressantes. En effet, on peut de cette manière déterminer une zone du cœur qui présente une anomalie ou détecter un infarctus.

Puis, on constate que le traitement de ce type de problème conduit naturellement à la résolution d'un problème de contrôle optimal. En effet, on cherche ici la solution qui minimise l'écart à la valeur du potentiel mesuré à la surface du cœur tout en respectant un modèle mathématique choisi pour l'activité cardiaque. En ce sens, on présente ici la solution d'un problème de contrôle optimal pour les équations monodomaines. On se limite au modèle ionique de Rogers-McCulloch présenté en 2.3.1.

6 Contrôle optimal pour les équations monodomaines

Le problème que l'on traite ici se rapporte à la défibrillation. On cherche à annuler l'effet d'une stimulation électrique sur le potentiel cardiaque dans un domaine qui présente une zone d'excitation en appliquant deux stimuli d'intensité optimale. Le problème que l'on traite est présenté dans [NK11], mais on ne suit pas une méthode de résolution similaire.

6.1 Présentation du problème

On considère un domaine carré $\Omega = [0, 5] \times [0, 5]$ dans lequel on applique en son centre un stimulus de 110 mV pour $t < 1$ ms. Si l'on laisse le système évoluer en l'état, le stimulus va exciter l'entier du domaine comme on l'a vu en 4.1. Ainsi, on veut appliquer deux stimuli placés en chaque centre des domaines $\Omega_{con1} = [1.5, 2] \times [2.25, 2.75]$ et $\Omega_{con2} = [3, 3.5] \times [2.25, 2.75]$ de rayons 0.25 et optimiser leur amplitude pour annuler l'effet du stimulus initial. Ces stimuli sont appliqués après l'instigation de l'excitation entre les instants $2 < t < 3$ ms. Ainsi, le problème peut s'écrire

$$I_e = \arg \min J(v, I_e) \quad (6.1)$$

sous la contrainte que le potentiel v satisfait les équations monodomaines, c.-à-d.

$$\begin{aligned} \left(\frac{\lambda}{1+\lambda} \right) \nabla \cdot (M_i \nabla v) &= C_m \chi \frac{\partial v}{\partial t} + \chi c_1 v(v-a)(1-v) - c_2 v w + I_{app} + I_e, \\ \frac{\partial w}{\partial t} &= b(v - c_3 w), \end{aligned} \quad (6.2)$$

où a, b, c_1, c_2, c_3 sont des paramètres donnés, voir Table 1. Le stimulus I_{app} représente le stimulus initial alors que I_e est le stimulus de défibrillation.

6.2 Dérivation du problème de contrôle optimal

Le problème donné par les équations (6.1) et (6.2) définissent un problème de contrôle optimal, où la fonctionnelle coût est donnée J et les variables d'état par v et w qui satisfont la contrainte (6.2).

Le but du problème étant de trouver le stimulus de défibrillation d'amplitude minimale qui permet d'annuler l'excitation initiale, on définit la fonctionnelle coût comme

$$J(v, I_e) = \frac{1}{2} \int_0^T \left(\int_{\Omega_{obs}} |v - v_0|^2 d\Omega_{obs} + \alpha \int_{control} |I_e|^2 d\Omega_{control} \right) dt, \quad (6.3)$$

où $\Omega_{control} = \Omega_{con1} \cup \Omega_{con2}$ et $\Omega_{obs} = \Omega \setminus \Omega_{control}$. Le paramètre α représente le coût de la variable de contrôle I_e et $v_0 = -84$ mV le potentiel au repos.

6.2.1 Lagrangienne du système

Pour résoudre le problème de contrôle, on suit la méthode des multiplicateurs de Lagrange, voir [Roz]. De cette manière, il nous faut définir la Lagrangienne du système :

$$\begin{aligned} \mathcal{L}(v, w; p, q; I_e) &= J(v, I_e) + \int_0^T \int_{\Omega} \left(\frac{\partial w}{\partial t} - g(v, w) \right) q d\Omega dt \\ &\quad + \int_0^T \int_{\Omega} \left(\nabla \cdot (\tilde{M}_i \nabla v) - \frac{\partial v}{\partial t} - I_{ion}(v, w) + I_{app} + I_e \right) p d\Omega dt, \end{aligned} \quad (6.4)$$

avec

$$\begin{aligned} I_{ion}(v, w) &= c_1 v(v-a)(1-v) - c_2 v w, \\ g(v, w) &= b(v - c_3 w). \end{aligned}$$

On se place ici dans un cas d'isotropie avec $C_m \chi = 1$.

6.2.2 Problème adjoint

Pour obtenir le problème adjoint, il nous faut dériver par rapport aux variables d'état la Lagrangienne (6.4). En annulant les dérivées partielles ainsi obtenues, on trouve la formulation faible du problème adjoint : trouver $(p, q) \in (\mathbf{H}^1(\Omega))^2$ tels que

$$\begin{aligned} 0 &= \frac{\partial \mathcal{L}}{\partial v}(\phi) = \int_0^T \int_{\Omega_{control}} (v - v_0) \phi d\Omega - \int_0^T \int_{\Omega} \frac{\partial g}{\partial v}(v, w) q \phi d\Omega dt \\ &\quad - \int_0^T \int_{\Omega} \left(\tilde{M}_i \nabla \cdot \phi \nabla p - \frac{\partial \phi}{\partial t} q - \frac{\partial I_{v,w}}{\partial v}(v, w) p \phi \right) d\Omega dt, \quad \forall \phi \in \mathbf{H}^1(\Omega), \\ 0 &= \frac{\partial \mathcal{L}}{\partial w}(\psi) = \int_0^T \int_{\Omega} \left(\frac{\partial \psi}{\partial t} q + \frac{\partial g}{\partial w}(v, w) q \psi \right) d\Omega dt \\ &\quad + \int_0^T \int_{\Omega} \frac{\partial I_{ion}}{\partial w}(v, w) p \psi d\Omega dt, \quad \forall \psi \in \mathbf{H}^1(\Omega), \end{aligned}$$

où (v, w) est la solution du problème d'état. Les conditions aux bords pour la variable adjointe p est identique à celle de la variable d'état v ; il s'agit d'une condition de Neumann homogène.

6.2.3 Condition d'optimalité

On obtient la condition d'optimalité en annulant la dérivée partielle de la Lagrangienne (6.4) par rapport à la variable de contrôle :

$$\delta J = 0 = \frac{\partial \mathcal{L}}{\partial I_e} = \alpha \int_0^T \int_{\Omega_{control}} I_e d\Omega_{control} dt + \int_0^T \int_{\Omega} I_e p d\Omega dt, \quad (6.5)$$

où p est solution du problème adjoint.

7 Résolution numérique du problème de contrôle optimal

On emploie le logiciel FreeFEM++ pour résoudre numériquement ce problème de contrôle optimal. On utilise pour chacun des domaines mentionnés précédemment les maillages non structurés automatiquement générés par le logiciel. Ceux-ci sont représentés à la Figure 15. Comme dans la partie précédente, on utilise exclusivement des éléments finis de degré 1.

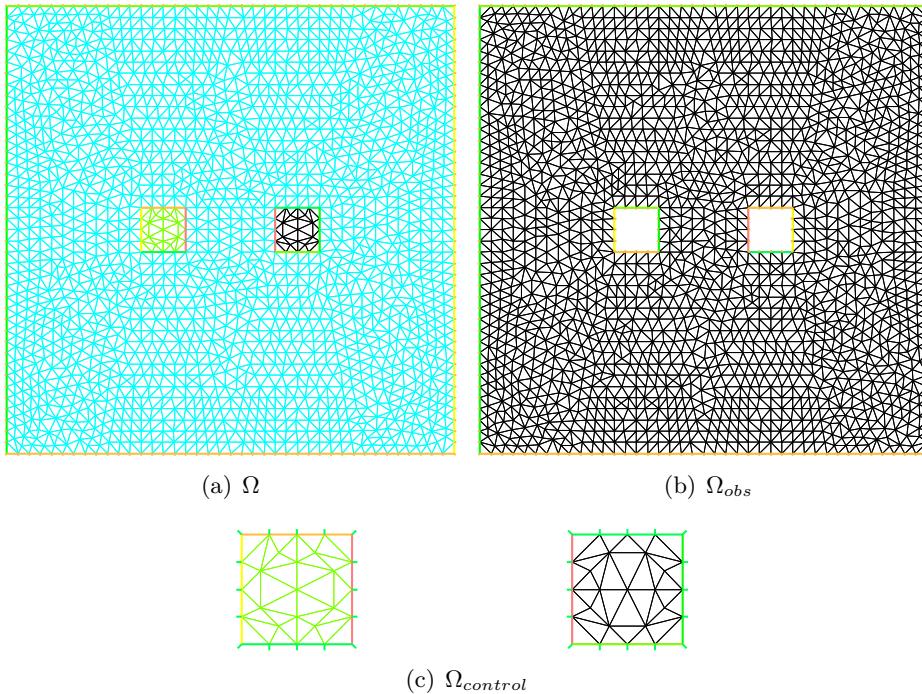


FIGURE 15 – Maillages

Afin de contrôler uniquement l'amplitude du stimulus de défibrillation, on définit celui-ci de la manière suivante :

$$I_e = \begin{cases} A & \text{si } ((x - 1.5)^2 + (y - 2.5)^2) < 0.25^2 \text{ ou } ((x - 3)^2 + (y - 3.5)^2) < 0.25^2, \\ 0 & \text{sinon.} \end{cases} \quad (7.1)$$

7.1 Boucle d'optimisation

On utilise la méthode de descente de la plus grande pente pour obtenir la valeur optimale de l'intensité du stimulus de défibrillation. Pour plus de détails sur la méthode, on peut se référer à [QSS07].

Problème d'état On commence par résoudre le problème d'état donné en (6.2). On utilise le schéma d'Euler rétrograde pour la dérivée temporelle. Les conditions initiales sont données par $v_0 = -84$ mV et $w_0 = 0$.

Problème adjoint Une fois la solution du problème d'état obtenue, on peut résoudre le problème adjoint. On le résout en temps inversé, avec les conditions finales $p = 0$ et $q = 0$.

Condition d'optimalité La solution du problème adjoint permet le calcul de la condition d'optimalité δJ . Le poids α de la variable de contrôle I_e est fixé à $\alpha = 10^{-3}$. Ce terme ne doit pas avoir trop d'influence sur la fonctionnelle coût mais peut permettre de contrôler l'intensité du stimulus.

Finalement, la mise à jour de la valeur A de l'amplitude du stimulus (7.1) est donnée par

$$A^{k+1} = A^k - \tau \delta J,$$

où δJ est donné par (6.5). Le pas d'itération τ est fixé et doit être choisi suffisamment petit pour garantir la convergence de l'algorithme.

La méthode complète de résolution est résumée par l'Algorithme 1.

Algorithme 1 Optimisation sous contraintes

```

 $A \leftarrow A_0, \delta J \leftarrow 0, \tau$  fixé ;
while  $\delta J >$  tolérance do
    Résolution du problème d'état  $\leftarrow (v, w)$  ;
    Résolution du problème adjoint sachant  $(v, w) \leftarrow (p, q)$  ;
    Calcul de la sensibilité  $\delta J$  ;
     $A \leftarrow A - \tau \delta J$  ;
end while
 $A_{opt} \leftarrow A$ .

```

Remarque 7.1

L'implémentation numérique de l'algorithme de résolution présenté précédemment ne donne pas des résultats satisfaisants dans le sens où il ne converge pas. Ceci provient peut-être du fait de l'ordre de discréétisation-optimisation du problème. Il peut ainsi être intéressant d'essayer l'autre méthode, mais ceci peut faire l'objet d'un développement futur et n'entre pas dans le cadre de ce projet.

7.2 Comportement physique du problème

Si nous n'avons pas encore à ce stade un algorithme de résolution convergeant, il est possible d'observer l'effet des stimuli de défibrillation sur un domaine excité et ainsi de donner un sens au problème que l'on traite. En ce sens, on présente l'effet de la défibrillation.

Le domaine est excité en début de simulation – $t < 1$ ms – par un stimulus central d'intensité 110 mV, voir Figure 16. Puis, on applique le stimulus de défrillation d'amplitude –1700 mV

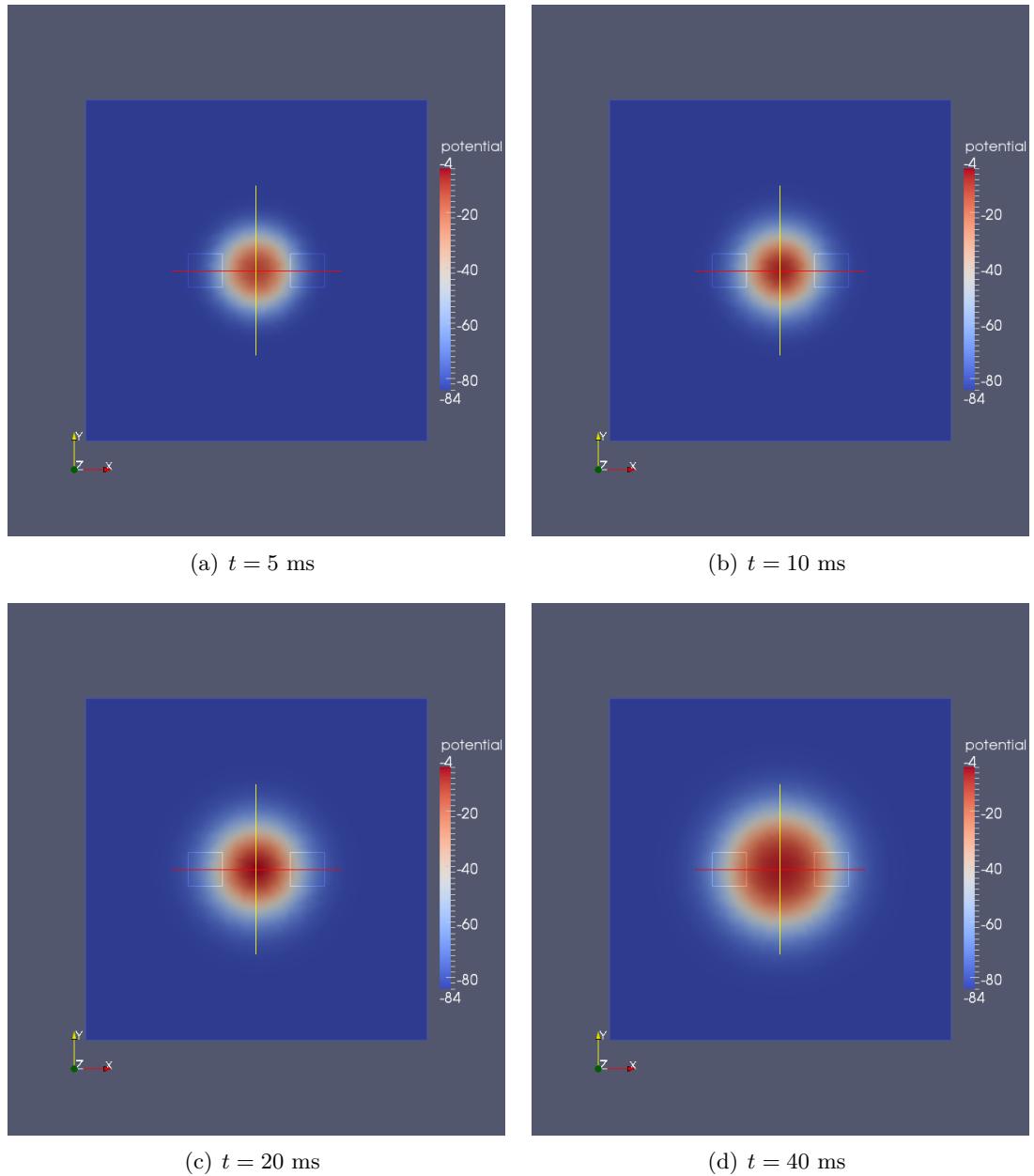


FIGURE 16 – Propagation de l'onde de potentiel donnée par le stimulus initial sans défibrillation.

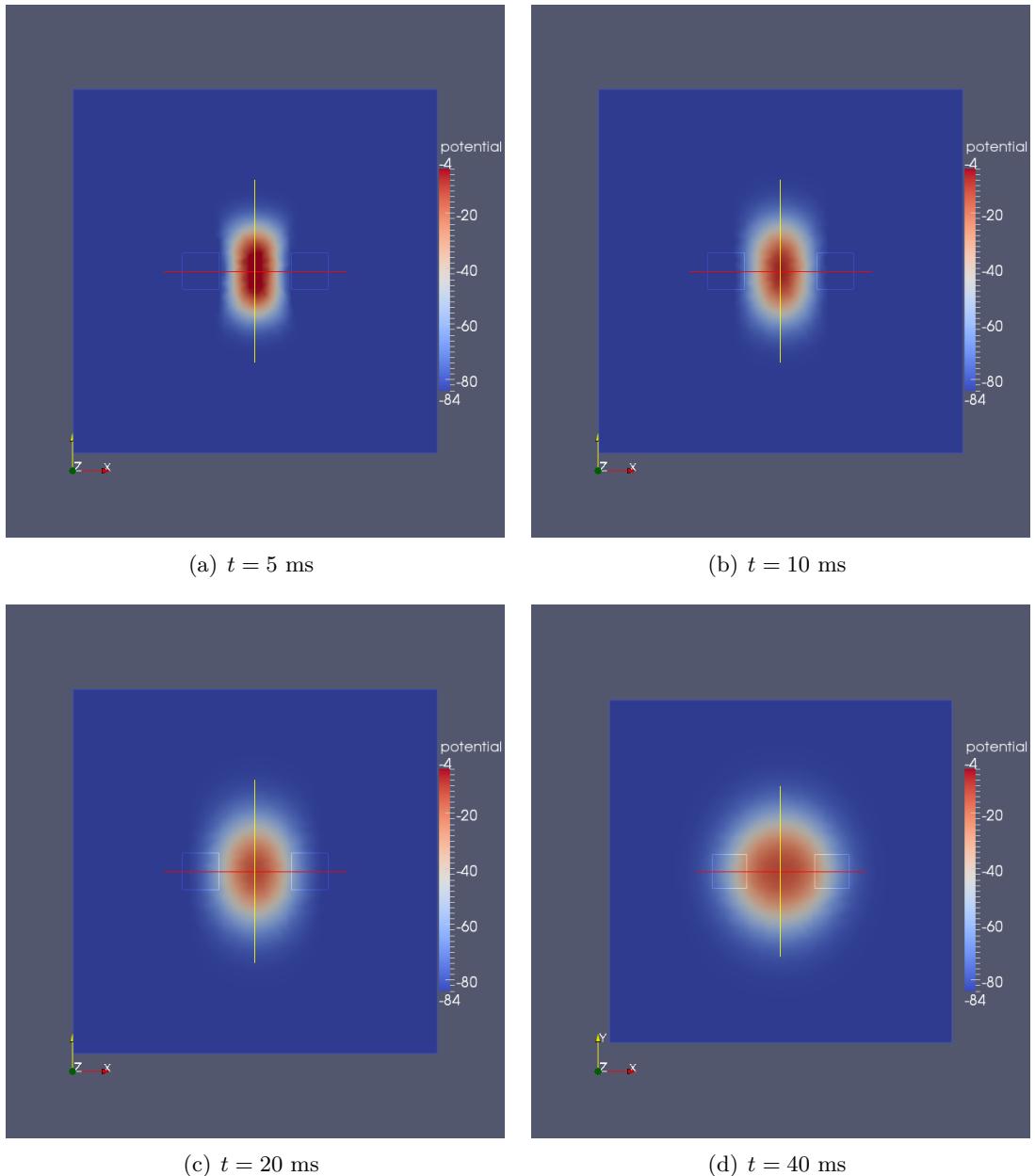


FIGURE 17 – Propagation de l'onde de potentiel donnée par le stimulus initial avec défibrillation.

entre les instants $0.5 < t < 2$ ms, voir Figure 16. L'amplitude maximale décroît d'environ 20 mV et la zone excitée réduit en taille.

Remarque 7.2

On a fait le choix ici d'optimiser l'amplitude du stimulus de défibrillation. Cependant, il peut être également judicieux d'optimiser le rayon de celui-ci. En effet, en multipliant son rayon par un facteur $\sqrt{2}$, on parvient annuler complètement l'effet de la stimulation initiale. On remarque que cette méthode est nettement plus efficace, voir Figure 18. Ceci porte à croire que la zone

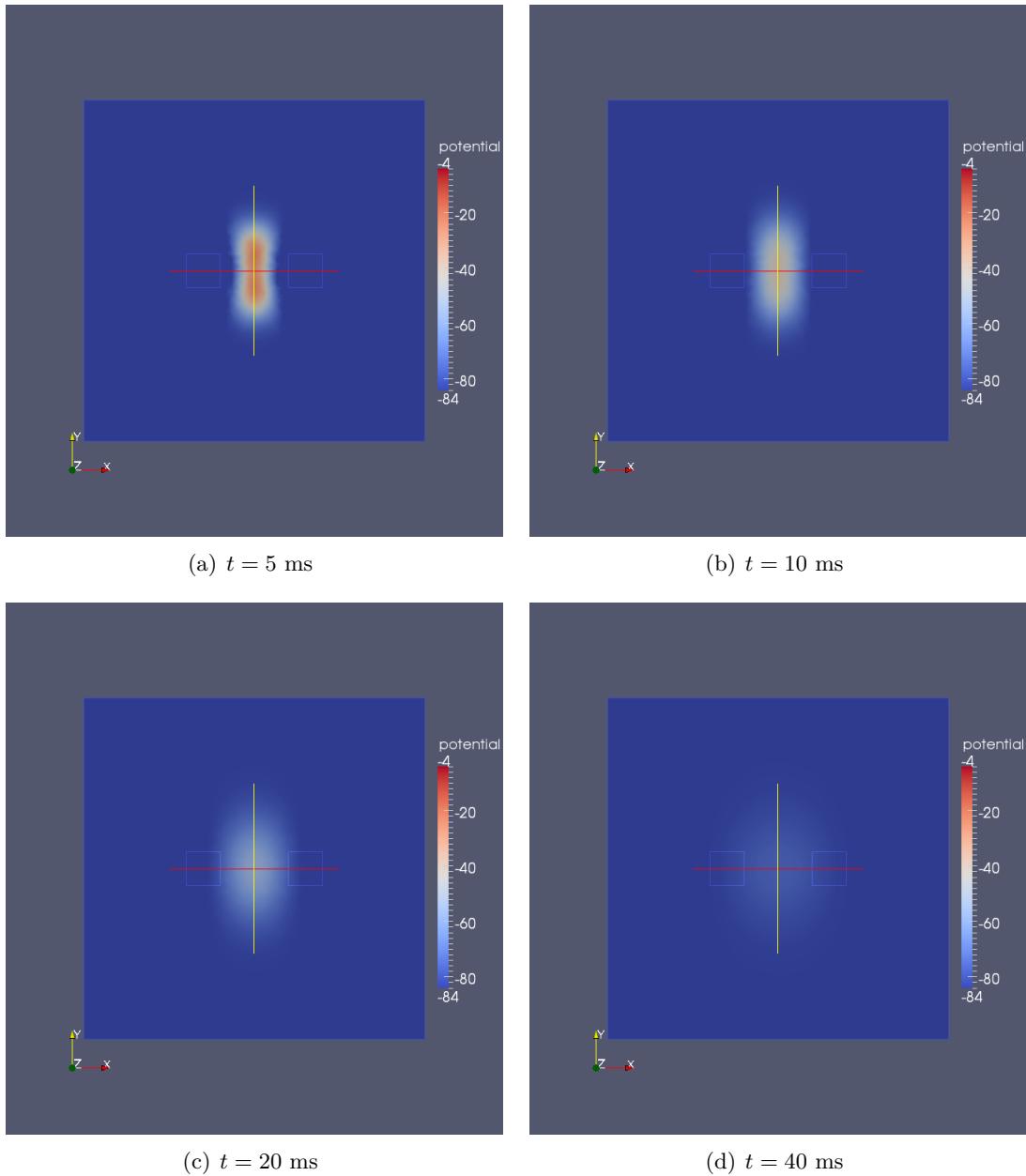


FIGURE 18 – Propagation de l'onde de potentiel donnée par le stimulus initial avec défibrillation (rayon du stimulus multiplié par un facteur $\sqrt{2}$).

de défibrillation que nous avons choisie initialement est trop petite et qu'ainsi on ne parvient pas à annuler l'effet du stimulus initial. En effet, en procédant à plusieurs simulations pour un

diamètre de défibrillation fixé, il semble qu'on atteint un seuil dans l'arrêt de la propagation du potentiel. Pour un diamètre fixé, on ne parvient pas à réduire l'excitation au delà d'un certain seuil. Pour un travail futur, il serait intéressant de chercher à la fois l'amplitude mais aussi l'aire d'application optimale du stimulus de défibrillation.

8 Conclusion

Ce travail a permis d'exposer différents modèles pour l'activité électrique dans le cœur et les oreillettes. On met en place les bases physiques nécessaires à leur compréhension, puis on s'attarde sur les modèles ioniques les plus couramment utilisés, où l'on met en avant les spécificités de chacun. On met l'accent sur le modèle CRN qui est au centre du projet. On présente également la discrétisation de modèles de ce type.

On atteint le but de ce travail en validant l'implémentation du code LifeV pour le modèle CRN. A partir de là, on peut comparer les différents modèles existants et prendre conscience de la diversité de ceux-ci. Chacun a sa spécificité et rend plus ou moins bien certaines caractéristiques du potentiel électrique dans les cellules cardiaques. De plus, on remarque que certains modèles sont spécifiques à certaines régions du cœur, ainsi il serait intéressant d'étudier le couplage de différents de ces modèles dans le but de modéliser l'activité du cœur dans son ensemble.

Finalement, ce travail ouvre la réflexion à certaines perspectives du contrôle optimal dans le domaine de l'électrophysiologie cardiaque. En effet, la défibrillation est au centre de la recherche du traitement des arythmies cardiaques. Ainsi, la poursuite de ce travail peut être de mener à bien l'implémentation de l'algorithme présenté en 6 puis de l'étendre aux équations bidomaines. Il serait également intéressant de pouvoir varier le modèle ionique utilisé.

Références

- [BOCF08] A. BUENO-OROVIO, E. M. CHERRY & F. H. FENTON – « Minimal model for human ventricular action potentials in tissue », *Journal of Theoretical Biology* **253** (2008), p. 544–560.
- [BR77] G. BEELER & H. REUTER – « Reconstruction of the action potential of myocardial ventricular fibres », *Journal of Physiology* (1977), no. 268, p. 177–210.
- [CFDE⁺06] P. COLLI FRANZONE, P. DEUFLHARD, B. ERDMANN, J. LANG & L. F. PAVARINO – « Adaptivity in space and time for reaction-diffusion systems in electrocardiology », *SIAM J. Sci. Comput.* **28** (2006), no. 3, p. 942–962.
- [CFS02] P. COLLI FRANZONE & G. SAVARÉ – « Degenerate evolution systems modeling the cardiac electric field at micro and macroscopic level », *Progress in nonlinear differential equations and their applications* **50** (2002), p. 49–78.
- [CRN98] M. COURTEMANCHE, R. J. RAMIREZ & S. NATTEL – « Ionic mechanisms underlying human atrial action potential properties : insights from a mathematical model », *AJP - Heart* **275** (1998), no. 1, p. H301–H321.
- [Fit61] R. FITZHUGH – « Impulses and physiological states in theoretical models of nerve membrane », *Biophys. J.* **1** (1961), p. 445–465.
- [GG08] L. GERARDO-GIORDA – « Modeling and numerical simulation of action potentials in human atrial tissues », *Technical Report, Emory University* (2008), no. TR 2008-004.
- [Hec09] F. HECHT – *FreeFEM++*, Laboratoire Jacques-Louis Lions, Université Pierre et Marie Curie, 3ème éd., 2009.
- [KS98] J. KEENER & J. SNEYD – *Mathematical physiology*, Springer, 1998.
- [Lif] « LifeV » – <http://www.lifev.org>.
- [LR91] C. LUO & Y. RUDY – « A model of the ventricular cardiac action potential : Depolarisation, repolarisation and their interaction », *Circulation Research* (1991), no. 68, p. 1501–1526.
- [MAT10] MATLAB – *version 7.10.0 (r2010a)*, The MathWorks Inc., Natick, Massachusetts, 2010.
- [NAY62] J. NAGUMO, S. ARIMOTO & S. YOSHIZAWA – « An active pulse transmission line simulating nerve axon », *Proceedings of the IRE* **50** (1962), no. 10, p. 2061–2070.
- [NK11] C. NAGAIAH & K. KUNISCH – « High order optimization and adaptive numerical solution for optimal control of monodomain equations in cardiac electrophysiology », *Applied Numerical Mathematics* **61** (2011), p. 53–65.
- [PCN⁺11] A. PULLAN, L. CHENG, M. NASH, D. BROOKES, A. GHODRATI & R. MACLEOD – « The inverse problem of electrocardiography », in *In Comprehensive electrocardiology : theory and practice in health and disease* (P. Macfarlane, A. van Oosterom, M. Janse, J. Camm, P. Kligfield & O. Pahlm, éds.), Springer, 2 éd., 2011, p. 299–344.
- [QSS07] A. QUARTERONI, R. SACCO & F. SALERI – *Méthodes numériques : algorithmes, analyse et applications*, Springer, 2007.
- [Qua09] A. QUARTERONI – *Numerical models for differential problems*, Springer, 2009.
- [RM94] J. ROGERS & A. McCULLOCH – « A colocation-Galerkin finite element model of cardiac action potential », *IEEE Trans. Biomed. Eng.* **41** (1994), p. 743–757.

- [Roz] G. ROZZA – *An overview on optimal control of partial differential equations*, Lecture notes.
- [SLC⁺06] J. SUNDNES, G. T. LINES, X. CAI, B. F. NIELSEN, K.-A. MARDAL & A. TVEITO – *Computing the electrical activity in the heart*, Springer, 2006.

Annexes

A Modèle RM

A.1 Code FreeFEM pour la résolution du modèle RM

```
1 load "iovtk";
2 //*****
3 // Diagonal conductivity tensor
4 real chi=1000, Cm=0.001, D=0.005,
5 D11 = D, D22=D, D12=0, D21=0;
6
7 // Ionic parameters
8 real a=0.13, b=0.013, c1=0.26, c2=0.1, c3=1.0, T=0.63, A=110, v0=-84.0, w0=0;
9
10 // Constant time step and final time
11 real dt= 0.5, Tfinal=600; int tfalse;
12
13 //Mesh
14 real s=5;
15 border Gamma1(t=0,s) {x=t; y=0;};
16 border Gamma2(t=0,s) {x=s; y=t; }
17 border Gamma3(t=0,s) {x=s-t; y=s;};
18 border Gamma4(t=0,s) {x=0; y=s-t; }
19 int mm=32;
20 mesh Th = buildmesh (Gamma1(mm)+ Gamma2(mm)
21 +Gamma3(mm)+ Gamma4(mm));
22
23
24 fespace Vh(Th,P1);
25 // potential, gating variable, pressure and scalar test function
26 Vh v=v0, vold, w=w0, wold, phi;
27
28 // source and reaction terms
29 Vh Iapp, Iion;
30
31 //*****
32 problem Monodomain(v,phi) = int2d(Th)(chi*Cm*v*phi/dt+D11*dx(v)*dx(phi)
33 +D12*dy(v)*dx(phi)+D21*dx(v)*dy(phi)+D22*dy(v)*dy(phi))
34 +int2d(Th)((chi*Cm*Iion-Iapp-chi*Cm*vold/dt)*phi);
35
36 //*****
37 for(real t=0;t<Tfinal;t+=dt){
38   tfalse=t;
39   cout << "t = " << t << endl;
40   vold=v; wold=w;
41   Iapp=A*((x^2 + y^2)<2)*(t<2)+((x-s*0.5)^2 + (y-s*0.5)^2<1)*(t>300 && t< 310));
42   Iion=c1/(T*A^2)*(vold-v0)*(vold-v0-a*A)*(vold-v0-A)+c2/T*(vold-v0)*wold;
43   Monodomain;
44   w=w+dt*b/(T*A)*(vold-v0-A*c3*wold);
45
46 //plot(v,value=true,fill=1);
47 savevtk("outputfiles/RMparameters/RMparameters"+tfalse+".vtk", Th, v, w, dataname="potential gating");
48
```

49 }

B Modèle CRN

B.1 Code Matlab pour la résolution du modèle ionique

Le code suivant permet la résolution du modèle ionique CRN grâce au logiciel Matlab. L'usage des figures que l'on peut générer ici permet, en comparaison des résultats de [GG08], la correction du code suivant B.2.

```
1 clear all;
2
3 %parameters
4 dt=0.005;
5
6 R=8.314472;
7 temperature=307.7532;
8 F=96.4853;
9 C=100.;

10
11 Ca0=1.8;
12 Na0=140. ;
13 K0=5.4;
14
15 Vi=13668. ;
16 Vup=1109.52;
17 Vrel=96.48;
18
19 KmNai=10. ;
20 KmK0=1.5;
21 KmNa=87.5;
22 KmCa=1.38;
23 Kup=0.00092;
24
25 consCaUpMax=15. ;
26 consTrpnMax=0.07;
27 consCmdnMax=0.05;
28 consCsqnMax=10. ;

29
30 KmTrpn=0.0005;
31 KmCmdn=0.00238;
32 KmCsqn=0.8;
33
34 gNa=7.8;
35 gKl=0.09;
36 gTo=0.1652;
37 gKr=0.0294;
38 gKs=0.129;
39 gCal=0.1238;
40 gbCa=0.00113;
41 gbNa=0.000674;
42
43 INaKmax=0.6;
44 INaCamax=1600. ;
45 IpCamax=0.275;
46 Iupmax=0.005;
```

```

47
48 krel=30.;
49
50 % initial values of gating variables
51 h=0.965;
52 d=1.37e-4;
53 xr=3.29e-5;
54 ai=0.999;
55 ui=0.999;
56 m=2.91e-3;
57 j=0.978;
58 f=0.999;
59 xs=1.87e-2;
60 aa=3.04e-2;
61 ua=4.96e-3;
62 fCa=0.775;
63 pv=1.;
64 pu=0.;
65 pw=0.999;
66
67 %initial concentrations
68 ConcentrationNa=11.2;
69 ConcentrationK=139.;
70 ConcentrationCa=1.02e-4;
71 ConcentrationCaUp=1.49;
72 ConcentrationCaRel=1.49;
73
74 %initialisation V, Fn
75 V=-82.;
76
77 % Na current
78 ah = 0.5*(1-sign(V+40+eps))*0.135.*exp(-1.0*(80+V)/6.8);
79 bh = (3.56*exp(0.079*V)+3.1*1e5*exp(0.35*V))*0.5.* (1-sign(V+40+eps)) + (1./(0.13*(1+exp
    (-1*(V+10.66)/11.1))))*0.5.* (1+sign(V+40+eps));
80 aj = 0.5*(1-sign(V+40+eps)).*(-1.2714*1e5*exp(0.2444*V) - 3.474*1e-5*exp(-0.04391*V))
    .*(V+37.78)./(1+exp(0.311*(V+79.23)));
81 bj = 0.5*(1-sign(V+40+eps)).*0.1212.*exp(-0.01052*V)./(1+exp(-0.1378*(V+40.14))) +
    0.5*(1+sign(V+40+eps))*0.3.*exp(-2.535*1e-7*V)./(1+exp(-0.1*(V+32)));
82 % if(-47.131<V<-47.129)
83 am = 0.32 * (V + 47.13) / (1. - exp(-0.1 * (V + 47.13)));
84 % else
85 %     am = 3.2;
86 % end
87 bm = 0.08*exp(-V/11.);
88
89 %transient outward K current
90 aaa = 0.65*1.0/(exp(-(V+10.0)/8.5)+ exp(-(V-30.0)/59.0));
91 baa=0.65*1.0/(2.5+exp((V+82.0)/17.0));
92
93 aai = 1.0/(18.53+exp((V+113.7)/10.95));
94 bai = 1.0/(35.56+exp(-(V+1.26)/7.44));
95
96 %ultra rapid delayed rectifier K current
97 aua=0.65*1.0/(exp(-(V+10.0)/8.5)+exp(-(V-30.0)/59.0));
98 bua=0.65*1.0/(2.5+exp((V+82.0)/17.0));

```

```

99
100 aui = 1.0/(21.0+exp(-(V-185.0)/28.0));
101 bui = exp((V-158.0)/16.0);
102
103 %rapid delayed rectifier outward K current
104 axr = 0.0003*(V+14.1)/(1.0-exp(-(V+14.1)/5.0));
105 bxr = 7.3898e-5*(V-3.3328)/(exp((V-3.3328)/5.1237)-1.0);
106
107 %slow delayed rectifier outward K current
108 axs = 4e-5*(V-19.9)/(1.0-exp(-(V-19.9)/17.0));
109 bxs = 3.5e-5*(V-19.9)/(exp((V-19.9)/9.0)-1.0);
110
111 for i=1:100000 %50000 iterations = 500 msec
112
113 %Stimulation current
114 Istim =0;
115 % current stimulation at t = 20 msec
116 if i==4000
117     Istim = - 10000.;%2800.0; %microamp/cm^2: stimulus current
118 end
119
120 %%%%%%%%%%%%%% membrane currents %%%%%%%%%%%%%%
121 % fast Na current
122 ENa = (R * temperature / F ) * log(Na0 / ConcentrationNa);
123 INa = gNa * m * m * m* h * j*(V-ENa);
124
125 % time independant K current
126 EK = (R * temperature / F ) * log(K0/ConcentrationK);
127 Klinf = 1.0/(1.0+exp(0.07*(V+80.0)));
128 IKl=gKl*Klinf*(V-EK);
129
130 % transient outward K current
131 Ito = gTo*aa*aa*aa*ai*(V-EK);
132
133 % ultra rapid delayed rectifier K current
134 gKur= 0.005 + 0.05/(1.0+exp(-(V-15.0)/13.0));
135 IKur = gKur*ua*ua*ua*ui*(V-EK);
136
137 % rapid delayed rectifier outward K current
138 IKr = gKr*xr*(V-EK)/(1.0+exp((V+15)/22.4));
139
140 % slow delayed rectifier outward K current
141 IKs = gKs*xs*xs*(V-EK);
142
143 % L-type Ca current
144 ICaL = gCaL*d*fCa*(V-65.0);
145
146 %Ca2+ pump current
147 Irel = krel *pu*pu*pv*pw*(ConcentrationCaRel-ConcentrationCa);
148
149 %~ Real satiga;
150 %~ satiga=
151 % Na-K pump
152 INaK = INaKmax*(1.0/(1.0+0.1245...
153             *exp(-0.1*V* F/(R * temperature) )...

```

```

154     +0.0365/7.0*(exp(Na0/67.3)-1.0)...  

155     *exp(-V* F/(R * temperature) )))...  

156     *(1.0/(1.0+power(KmNai/ConcentrationNa,1.5)))...  

157     *(K0/(K0+KmK0));  

158  

159 % Na-Ca exchanger  

160 INaCa = INaCamax/((power(KmNa,3.0)+power(Na0,3.0))...  

161     *(KmCa+Ca0)*(1.0+0.1*exp((0.35-1.0)*V...  

162     * F/(R * temperature)))*(exp(0.35*V* F...  

163     /(R * temperature) )*power(ConcentrationNa,3.0)...  

164     * Ca0 - exp((0.35-1.0)*V* F/(R * temperature))...  

165     *power(Na0,3.0)*ConcentrationCa));  

166  

167 % background current  

168 ECan=(R * temperature)/(2.*F)*log(Ca0/ConcentrationCa); % modif retrait *0.5  

169 ICab = gbCa*(V-ECan);  

170 % ENa=(R * temperature)*log(Na0/ConcentrationNa);  

171 INab = gbNa*(V-ENa);  

172  

173 % Ca pump  

174 IpCa = IpCamax*(ConcentrationCa)/(0.0005 + ConcentrationCa);  

175  

176 % Sarcoplasmic currents  

177  

178 % Ca release current from JSR  

179  

180 %%%%%%%%%%%%%% MODIF %%%%%%%%%%%%%%  

181 % CurrentIrel= krel*pu*pu*pv*pw*(ConcentrationCaRel-ConcentrationCa);  

182 Fn= Vrel * Irel - (5.0 * 1e-1 / F) * (0.5 * ICaL - 0.2 * INaCa);  

183  

184 % transfert current from NSR to JSR  

185 Itr =(ConcentrationCaUp -ConcentrationCaRel)/180.0;  

186  

187 % Ca uptake current by the NSR  

188 Iup = Iupmax*ConcentrationCa/(ConcentrationCa+Kup); %% !!!  

189 % Ca leak current by the NSR  

190 Iupleak = Iupmax*ConcentrationCaUp/consCaUpMax;  

191  

192 %%%%%%%%%%%%%%  

193 % The original model from the CRN paper contains buffer modeling. They were not  

194 % implemented here because they have no incidence on transmembrane current.  

195 %%%%%%%%%%%%%%  

196  

197 INatot = 3.0 * INaK + 3.0 * INaCa + INab + INa;  

198 IKtot = -2.0 * INaK + IK1 + Ito + IKur + IKr + IKs;  

199 ICatot = -2.0 * INaCa + IpCa + ICaL + ICab;  

200  

201 Iion= INatot + IKtot + ICatot;  

202 % ionicCurrent(i)=Iion;  

203  

204 % keyboard  

205  

206 % for homogeneity reasons the transmembrane currents IKtot, INatot and ICatot  

207 % are multiplied by C (this multiplication doesn't appear in the original

```

```

209 % paper), but the sarcoplasmic currents are not multiplied
210
211 %%%%%% Tau et Inf %%%%%%
212 %%%%%%
213 %%%%%%
214 hinf = ah / (ah + bh);
215 tauh = 1. / (ah + bh);
216 jinf = aj / (aj + bj);
217 tauj = 1. / (aj + bj);
218 minf = am / (am + bm);
219 taum = 1. / (am + bm);
220
221 tauaa=1.0/((aaa+baa)*3.0);
222 aainf = 1.0/(1.0+exp(-(V+20.47)/17.54));
223
224 tauai=1.0/((aai+bai)*3.0);
225 aiinf = 1.0/(1.0+exp((V+43.1)/5.3));
226
227 tauua=(1.0/(aua+bua))/3.0;
228 uainf = 1.0/(1.0+exp(-(V+30.3)/9.6));
229
230 tauui=(1.0/(aui+bui))/3.0;
231 uiinf = 1.0/(1.0+exp((V-99.45)/27.48));
232
233 tauxr=1.0/(axr+bxr);
234 xrinf = 1.0/(1.0+exp(-(V+14.1)/6.5));
235 tauxs=0.5/(axs+bxs);
236 xsinf = power((1.0+exp(-(V-19.9)/12.7)),(-1.0/2.0));
237
238 %L-type Ca current
239
240 taud = (1.0-exp(-(V+10.0)/6.24))/(0.035*(V+10.0)*(1.0+exp(-(V+10.0)/6.24)));
241 dinf=1.0/(1.0+exp(-(V+10.0)/8.0));
242
243 tauf=9.0*1.0/(0.0197*exp(-power(0.0337,2.0)*power((V+10.0),2.0))+0.02);
244 finf=1.0/(1.0+exp((V+28.0)/6.9));
245
246 taufca= 2.0;
247 fCainf=1.0/(1.0+(ConcentrationCa/0.00035));
248
249 %Ca release current from JSR
250 %Fn expression has been simplified, being equivalent as the initial expression (
251 % simplification of the powerer of 10, in the
252 %following expressions using Fn). Furthermore, a modification has been done :
253 % instead of 10^-13, we put 10^-10
254
255 taupu=8.0;
256 puinf=1.0/(1.0+exp(-(Fn*power(10,4)-3.4175*power(10,3))/(13.67)));
257
258 taupv = 1.91 + 2.09 / (1.0 + exp(-(Fn * power(10, 4.0) - 3.4175 * power(10, 3)) /
259 pvinf = 1.0 - 1.0 / (1.0 + exp(-(Fn * power(10, 4.0) - 6.835 * power(10, 2)) /
taupw=6.0*(1.0-exp(-(V-7.9)/5.0))/((1.0+0.3*exp(-(V-7.9)/5.0))*(V-7.9));

```

```

260 pwinfof=1.0-1.0/(1.0 + exp(-(V-40.0)/17.0));
261
262 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% update V and gating variables
263
264 deltaV= -Iion-Istim;
265
266 V=V+dt*deltaV;
267
268 h=hinfof+(h-hinfof)*exp(-dt/tauh);
269 j=jinfof+(j-jinfof)*exp(-dt/tauj);
270 m=minfof+(m-minfof)*exp(-dt/taum);
271 aa=aainfof+(aa-aainfof)*exp(-dt/tauaa);
272 ai=aiinfof+(ai-aiinfof)*exp(-dt/tauai);
273 ua=uainfof+(ua-uainfof)*exp(-dt/tauua);
274 ui=uiinfof+(ui-uiinfof)*exp(-dt/tauui);
275 xr=xrinfof+(xr-xrinfof)*exp(-dt/tauxr);
276 xs=xsinfof+(xs-xsinfof)*exp(-dt/tauxs);
277 d=dinfof+(d-dinfof)*exp(-dt/taud);
278 f=finfof+(f-finfof)*exp(-dt/tauf);
279 fCa=fCainfof+(fCa-fCainfof)*exp(-dt/taufca);
280 pu=puinfof+(pu-puinfof)*exp(-dt/taupu);
281 pv=pvinfof+(pv-pvinfof)*exp(-dt/taupv);
282 pw=pwinfof+(pw-pwinfof)*exp(-dt/taupw);
283
284 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
285 %% Update ionic concentration
286
287 IonicChangeK= -IKtot / (F*Vi);%C *
288 IonicChangeNa=-INatot / (F*Vi); %C *
289 IonicChangeCa=(-ICatot / (2.0 * F * Vi)+(Vup *... %C *
290 (Iupleak - Iup)+ Irel * Vrel) / Vi) / (1.0 +...
291 consTrpnMax * KmTrpn / ( ConcentrationCa + KmTrpn).^2 ...
292 + consCmdnMax * KmCmdn / (ConcentrationCa + KmCmdn).^2);
293 IonicChangeCaUp=Iup-Iupleak-Itr*Vrel/Vup;
294 IonicChangeCaRel=(Itr-Irel)/(1.0+consCsqnMax*KmCsqn...
295 /power(ConcentrationCaRel+KmCsqn,2.0));
296
297
298
299
300 ConcentrationK=dt*IonicChangeK+ConcentrationK;
301 ConcentrationNa=dt*IonicChangeNa+ConcentrationNa;
302 ConcentrationCa=dt*IonicChangeCa+ConcentrationCa;
303 ConcentrationCaUp=dt*IonicChangeCaUp+ConcentrationCaUp;
304 ConcentrationCaRel=dt*IonicChangeCaRel+ConcentrationCaRel;
305
306
307 % Na current
308
309 ah = 0.5*(1-sign(V+40+eps))*0.135.*exp(-1.0*(80+V)/6.8);
310 bh = (3.56*exp(0.079*V)+3.1*1e5*exp(0.35*V))*0.5.* (1-sign(V+40+eps))+
311 (1./ (0.13*(1+exp(-1*(V+10.66)/11.1))))*0.5.* (1+sign(V+40+eps));
312 aj = 0.5*(1-sign(V+40+eps)).*(-1.2714*1e5*exp(0.2444*V) - 3.474*1e-5*exp(-0.04391*
313 V)).*(V+37.78)./(1+exp(0.311*(V+79.23)));

```

```

313     bj = 0.5*(1-sign(V+40+eps)).*0.1212.*exp(-0.01052*V)./(1+exp(-0.1378*(V+40.14))) +
      0.5*(1+sign(V+40+eps))*0.3.*exp(-2.535*1e-7*V)./(1+exp(-0.1*(V+32)));
314
315     am = 0.32 * (V + 47.13) / (1. - exp(-0.1 * (V + 47.13)));
316     bm = 0.08*exp(-V/11.);
317
318 %transient outward K current
319 aaa = 0.65*1.0/(exp(-(V+10.0)/8.5)+ exp(-(V-30.0)/59.0));
320 baa=0.65*1.0/(2.5+exp((V+82.0)/17.0));
321
322 aai = 1.0/(18.53+exp((V+113.7)/10.95));
323 bai = 1.0/(35.56+exp(-(V+1.26)/7.44));
324
325 %ultra rapid delayed rectifier K current
326 aua=0.65*1.0/(exp(-(V+10.0)/8.5)+exp(-(V-30.0)/59.0));
327 bua=0.65*1.0/(2.5+exp((V+82.0)/17.0));
328
329 aui = 1.0/(21.0+exp(-(V-185.0)/28.0));
330 bui = exp((V-158.0)/16.0);
331
332 %rapid delayed rectifier outward K current
333 axr = 0.0003*(V+14.1)/(1.0-exp(-(V+14.1)/5.0));
334 bxr = 7.3898e-5*(V-3.3328)/(exp((V-3.3328)/5.1237)-1.0);
335
336 %slow delayed rectifier outward K current
337 axs = 4e-5*(V-19.9)/(1.0-exp(-(V-19.9)/17.0));
338 bxs = 3.5e-5*(V-19.9)/(exp((V-19.9)/9.0)-1.0);
339
340 % %Recording the time series of the voltage and currents
341 % rec_Na(i)=ConcentrationNa;
342 % rec_Ca(i)=ConcentrationCa;
343 % rec_K(i)=ConcentrationK;
344 % rec_CaRel(i)=ConcentrationCaRel;
345 % rec_CaUp(i)=ConcentrationCaUp;
346 rec_V(i)=V;
347
348 %Recording evolution of gation variables
349 % rec_m(i)=m;
350 % rec_h(i)=h;
351 % rec_xs(i)=xs;
352 % rec_d(i)=d;
353 % rec_j(i)=j;
354 % rec_ua(i)=ua;
355 % rec_f(i)=f;
356 % rec_fCa(i)=fCa;
357 % rec_ui(i)=ui;
358 % rec_aa(i)=aa;
359 % rec_pu(i)=pu;
360 % rec_pv(i)=pv;
361 % rec_ai(i)=ai;
362 % rec_xr(i)=xr;
363 % rec_pw(i)=pw;
364 end
365
366

```

```

367 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
368 %%%%%% Post Processing %%%%%%
369 %%%%%%
370
371 l=linspace(0,10e4,10e4)*0.005;
372
373 figure(1)
374 plot(l,rec_V)
375 xlabel('time'), ylabel('V')
376
377 % figure(1)
378 % title('Transmembrane potential and ionic concentrations')
379 % subplot(321), plot(l,rec_V)
380 % xlabel('time'), ylabel('V')
381 % subplot(322), plot(l,rec_Na)
382 % xlabel('time'), ylabel('[Na]i'), ylim([11.199,11.201])
383 % subplot(323), plot(l,rec_K)
384 % xlabel('time'), ylabel('[K]i'), ylim([138.999,139.001])
385 % subplot(324), plot(l,rec_Ca)
386 % xlabel('time'), ylabel('[Ca]i'), ylim([1.e-4,1.04e-4])
387 % subplot(325), plot(l,rec_CaUp)
388 % xlabel('time'), ylabel('[Ca]up}')
389 % subplot(326), plot(l,rec_CaRel)
390 % xlabel('time'), ylabel('[Ca]rel')

391
392 % figure(1)
393 % title('Gating variables')
394 % subplot(4,2,1), plot(l,rec_m)
395 % xlabel('time'), ylabel('m')
396 % subplot(4,2,2), plot(l,rec_h)
397 % xlabel('time'), ylabel('h')
398 % subplot(4,2,3), plot(l,rec_xs)
399 % xlabel('time'), ylabel('xs')
400 % subplot(4,2,4), plot(l,rec_d)
401 % xlabel('time'), ylabel('d')
402 % subplot(4,2,5), plot(l,rec_j)
403 % xlabel('time'), ylabel('j')
404 % subplot(4,2,6), plot(l,rec_ua)
405 % xlabel('time'), ylabel('ua')
406 % subplot(4,2,7), plot(l,rec_f)
407 % xlabel('time'), ylabel('f')
408 % subplot(4,2,8), plot(l,rec_fCa)
409 % xlabel('time'), ylabel('fCa')
410 %
411 % figure(2)
412 % subplot(4,2,1), plot(l,rec_ui)
413 % xlabel('time'), ylabel('ui')
414 % subplot(4,2,2), plot(l,rec_aa)
415 % xlabel('time'), ylabel('aa')
416 % subplot(4,2,3), plot(l,rec_pu)
417 % xlabel('time'), ylabel('pu')
418 % subplot(4,2,4), plot(l,rec_pv)
419 % xlabel('time'), ylabel('pv')
420 % subplot(4,2,5), plot(l,rec_ai)
421 % xlabel('time'), ylabel('ai')

```

```

422 % subplot(4,2,6), plot(l,rec_xr)
423 % xlabel('time'), ylabel('xr')
424 % subplot(4,2,7), plot(l,rec_pw)
425 % xlabel('time'), ylabel('pw')

```

B.2 Corrections du modèle CRN de LifeV

Le code qui suit est un extrait du code `HeartIonicSolver.hpp` qui ne concerne que le modèle CRN. Un commentaire indique chaque modification effectuée.

```

1 //////////////////////////////////////////////////////////////////
2 //Model for the electrophysiology in the left atrium.
3 //Parameters and methods as in Courtemanche-Ramirez-Nattel 98.
4 //////////////////////////////////////////////////////////////////
5
6
7 template< typename Mesh,
8         typename SolverType = LifeV::SolverAztecOO >
9 class CourtemancheRamirezNattel : public virtual HeartIonicSolver<Mesh, SolverType>
10 {
11 public:
12     typedef typename HeartIonicSolver<Mesh, SolverType>::data_Type data_Type;
13     typedef typename HeartIonicSolver<Mesh, SolverType>::vector_Type vector_Type;
14     typedef typename HeartIonicSolver<Mesh, SolverType>::function_Type function_Type;
15
16     CourtemancheRamirezNattel( const data_Type&      dataType,
17                             const Mesh&          mesh,
18                             FESpace<Mesh, MapEpetra>& uFESpace,
19                             Epetra_Comm&           comm );
20
21     virtual ~CourtemancheRamirezNattel() {}
22
23     void updateRepeated( );
24
25     void updateElementSolution( UInt eleID );
26
27     void computeODECoefficients( const Real& u_ig, const Int& ig );
28
29     void solveIonicModel( const vector_Type& u, const Real timeStep );
30
31     void computeIonicCurrent( Real Capacitance,
32                           VectorElemental& elvec,
33                           VectorElemental& elvec_u,
34                           FESpace<Mesh, MapEpetra>& uFESpace );
35
36
37     void initialize ( );
38
39     //!< Returns the local solution vector for each field
40     const vector_Type& solutionGatingH() const {return M_solutionGatingH;}
41     const vector_Type& solutionGatingJ() const {return M_solutionGatingJ;}
42     const vector_Type& solutionGatingM() const {return M_solutionGatingM;}
43     const vector_Type& solutionGatingAA() const {return M_solutionGatingAA;}
44     const vector_Type& solutionGatingAI() const {return M_solutionGatingAI;}
45     const vector_Type& solutionGatingUA() const {return M_solutionGatingUA;}

```

```

46     const vector_Type& solutionGatingUI() const {return M_solutionGatingUI;}
47     const vector_Type& solutionGatingXR() const {return M_solutionGatingXR;}
48     const vector_Type& solutionGatingXS() const {return M_solutionGatingXS;}
49     const vector_Type& solutionGatingD() const {return M_solutionGatingD;}
50     const vector_Type& solutionGatingF() const {return M_solutionGatingF;}
51     const vector_Type& solutionGatingFCa() const {return M_solutionGatingFCa;}
52     const vector_Type& solutionGatingPU() const {return M_solutionGatingPU;}
53     const vector_Type& solutionGatingPV() const {return M_solutionGatingPV;}
54     const vector_Type& solutionGatingPW() const {return M_solutionGatingPW;}
55     const vector_Type& vectorConcentrationNa() const {return M_vectorConcentrationNa;}
56     const vector_Type& vectorConcentrationK() const {return M_vectorConcentrationK;}
57     const vector_Type& vectorConcentrationCa() const {return M_vectorConcentrationCa;}
58     const vector_Type& vectorConcentrationCaRel() const {return
      M_vectorConcentrationCaRel;}
59     const vector_Type& vectorConcentrationCaUp() const {return
      M_vectorConcentrationCaUp;}
60
61     Real M_R, M_temperature, M_F, M_permeabilityRatio, M_C, M_Ca0, M_Na0, M_K0,
62     M_Vi, M_Vup, M_Vrel, M_KmNai, M_KmKO, M_KmNa, M_KmCa, M_Kup, M_consCaUpMax,
63     M_consTrpnMax, M_consCmdnMax, M_consCsqnMax, M_KmTrpn, M_KmCmdn,
64     M_KmCsqn, M_gNa, M_gKl, M_gTo, M_gKr, M_gKs, M_gCal, M_gbCa, M_gbNa,
65     M_INaKmax, M_INaCamax, M_IpcCamax, M_Iupmax, M_krel, M_Irel,
66     M_ah, M_aj, M_bh, M_bj, M_am, M_bm, M_tauh, M_tauj, M_taum,
67     M_hinf, M_jinf, M_minf, M_akl, M_bkl, M_aaa, M_baa, M_aai, M_bai,
68     M_tauai, M_ainf, M_tauaa, M_aainf, M_aua, M_bua, M_aui, M_bui, M_tauui,
69     M_uainf, M_tauua, M_uainf, M_axr, M_bxr, M_tauxr, M_xrinf, M_axs, M_bxs,
70     M_tauxs, M_xsinf, M_dinf, M_finf, M_taud, M_tauf, M_fCainf, M_taufca, M_taupu,
71     M_puinf, M_taupv, M_pvinf, M_taupw, M_pwinf, M_EK, M_EKl, M_ENa, M_Klinf,
      M_ECan, M_gKur;
72
73     // Na current
74     Real M_INa;
75     // Potassium current
76     Real M_IKl;
77     // K transient current
78     Real M_Ito;
79     // K ultra rapid current
80     Real M_IKur;
81     // K rapid current
82     Real M_IKr;
83     // K slow current
84     Real M_IKs;
85     // Ca current
86     Real M_ICal;
87     // Na-K exchange
88     Real M_INaK;
89     // Na-Ca exchange
90     Real M_INaCa;
91     // Ca backward
92     Real M_ICab;
93     // Na backward
94     Real M_INab;
95     // Ca pump
96     Real M_IpCa;
97

```

```

98 // Ca transfert into the reticulum
99  Real M_Itr;
100 // Ca transfert into the reticulum from the cell
101  Real M_Iup;
102 // leak of Ca
103  Real M_Iupleak;
104
105 // Total Na currents
106 Real M_INatot;
107 // Total K currents
108  Real M_IKtot;
109 // Total Ca currents
110  Real M_ICatot;
111
112
113 vector_Type M_vectorExponentialh;
114 vector_Type M_vectorExponentialj;
115 vector_Type M_vectorExponentialm;
116 vector_Type M_vectorExponentialaa;
117 vector_Type M_vectorExponentialai;
118 vector_Type M_vectorExponentialua;
119 vector_Type M_vectorExponentialui;
120 vector_Type M_vectorExponentialxr;
121 vector_Type M_vectorExponentialxs;
122 vector_Type M_vectorExponentiald;
123 vector_Type M_vectorExponentialf;
124 vector_Type M_vectorExponentialfca;
125 vector_Type M_vectorExponentialpu;
126 vector_Type M_vectorExponentialpv;
127 vector_Type M_vectorExponentialpw;
128 vector_Type M_vectorInfimumh;
129 vector_Type M_vectorInfimumj;
130 vector_Type M_vectorInfimumm;
131 vector_Type M_vectorInfimumaa;
132 vector_Type M_vectorInfimumai;
133 vector_Type M_vectorInfimumua;
134 vector_Type M_vectorInfimumui;
135 vector_Type M_vectorInfimumxr;
136 vector_Type M_vectorInfimumxs;
137 vector_Type M_vectorInfimumd;
138 vector_Type M_vectorInfimumf;
139 vector_Type M_vectorInfimumfca;
140 vector_Type M_vectorInfimumpu;
141 vector_Type M_vectorInfimumpv;
142 vector_Type M_vectorInfimumpw;
143 vector_Type M_vectorFn;
144 vector_Type M_vectorIonicChange;
145
146 protected:
147 //! Global solution h
148 vector_Type M_solutionGatingH;
149 //! Global solution j
150 vector_Type M_solutionGatingJ;
151 //! Global solution m
152 vector_Type M_solutionGatingM;

```

```

153 //!Global solution AA
154     vector_Type           M_solutionGatingAA;
155 //!Global solution AI
156     vector_Type           M_solutionGatingAI;
157 //!Global solution UA
158     vector_Type           M_solutionGatingUA;
159 //!Global solution UI
160     vector_Type           M_solutionGatingUI;
161 //!Global solution XR
162     vector_Type           M_solutionGatingXR;
163 //!Global solution XS
164     vector_Type           M_solutionGatingXS;
165 //! Global solution d
166     vector_Type           M_solutionGatingD;
167 //! Global solution f
168     vector_Type           M_solutionGatingF;
169 //!Global solution FCa
170     vector_Type           M_solutionGatingFCa;
171 //!Global solution PU
172     vector_Type           M_solutionGatingPU;
173 //!Global solution PV
174     vector_Type           M_solutionGatingPV;
175 //!Global solution PW
176     vector_Type           M_solutionGatingPW;
177 //!Global solution Na_i
178     vector_Type           M_vectorConcentrationNa;
179 //!Global solution K_i
180     vector_Type           M_vectorConcentrationK;
181 //! Global solution Ca_i
182     vector_Type           M_vectorConcentrationCa;
183 //!Global solution Ca_rel
184     vector_Type           M_vectorConcentrationCaRel;
185 //!Global solution Ca_up
186     vector_Type           M_vectorConcentrationCaUp;
187
188     vector_Type   M_vectorIonicChangeCa;
189     vector_Type   M_vectorIonicChangeNa;
190     vector_Type   M_vectorIonicChangeK;
191     vector_Type   M_vectorIonicChangeCaUp;
192     vector_Type   M_vectorIonicChangeCaRel;
193 // Release of Ca from the reticulum
194     vector_Type   M_vectorCurrentIrel;
195     vector_Type   M_ionicCurrent;
196     vector_Type   M_ionicCurrentRepeated;
197     VectorElemental M_elemVecIonicCurrent;
198
199 private:
200 };
201
202
203 //
204 // IMPLEMENTATION
205 //
206
207

```

```

208  ///! Constructor
209  template<typename Mesh, typename SolverType>
210  CourtemancheRamirezNattel<Mesh, SolverType>::
211  CourtemancheRamirezNattel( const data_Type& dataType,
212    const Mesh& mesh,
213    FESpace<Mesh, MapEpetra>& uFEspace,
214    Epetra_Comm& comm ):
215    HeartIonicSolver<Mesh, SolverType>( dataType, mesh, uFEspace, comm ),
216    M_R(8.314472), //%% Joules/(Kelvin*mole)
217    M_temperature(307.7532), //%% kelvins
218    M_F(96.48533838), //%% coulombs/mmole
219    //M_permeabilityRatio(0.01833), //%% Na/K permeability ratio //md 04.2012
220    M_C(100.), //%% membrane capacitance set as 1 p-F/cm^2
221    // Concentration constants
222    M_Ca0(1.8),
223    M_Na0(140.),
224    M_K0(5.4),
225    // volume constants
226    M_Vi(13668),
227    M_Vup(1109.52),
228    M_Vrel(96.48),
229
230    M_KmNai(10.),
231    M_KmK0(1.5),
232    M_KmNa(87.5),
233    M_KmCa(1.38),
234    M_Kup(0.00092),
235
236    M_consCaUpMax(15),
237    M_consTrpnMax(0.07),
238    M_consCcmdnMax(0.05),
239    M_consCsqnMax(10),
240
241    M_KmTrpn(0.0005),
242    M_KmCcmdn(0.00238),
243    M_KmCsqn(0.8),
244
245    // model constants
246    M_gNa(7.8),
247    M_gK1(0.09),
248    M_gTo(0.1652),
249    M_gKr(0.0294),
250    M_gKs(0.129),
251    M_gCal(0.1238),
252    M_gbCa(0.00113),
253    M_gbNa(0.000674),
254    M_INaKmax(0.6),
255    M_INaCamax(1600),
256    M_IpCamax(0.275),
257    M_Iupmax(0.005),
258    M_krel(30),
259
260    // gating variables
261    M_vectorExponentialh(HeartIonicSolver<Mesh,SolverType>::M_localMap),
262    M_vectorExponentialj(HeartIonicSolver<Mesh,SolverType>::M_localMap),

```

```

263 M_vectorExponentialm(HeartIonicSolver<Mesh, SolverType>::M_localMap),
264 M_vectorExponentialaa(HeartIonicSolver<Mesh, SolverType>::M_localMap),
265 M_vectorExponentialai(HeartIonicSolver<Mesh, SolverType>::M_localMap),
266 M_vectorExponentialua(HeartIonicSolver<Mesh, SolverType>::M_localMap),
267 M_vectorExponentialui(HeartIonicSolver<Mesh, SolverType>::M_localMap),
268 M_vectorExponentialxr(HeartIonicSolver<Mesh, SolverType>::M_localMap),
269 M_vectorExponentialxs(HeartIonicSolver<Mesh, SolverType>::M_localMap),
270 M_vectorExponentialid(HeartIonicSolver<Mesh, SolverType>::M_localMap),
271 M_vectorExponentiallf(HeartIonicSolver<Mesh, SolverType>::M_localMap),
272 M_vectorExponentialfca(HeartIonicSolver<Mesh, SolverType>::M_localMap),
273 M_vectorExponentialpu(HeartIonicSolver<Mesh, SolverType>::M_localMap),
274 M_vectorExponentialpv(HeartIonicSolver<Mesh, SolverType>::M_localMap),
275 M_vectorExponentialpw(HeartIonicSolver<Mesh, SolverType>::M_localMap),
276 M_vectorInfimumh(HeartIonicSolver<Mesh, SolverType>::M_localMap),
277 M_vectorInfimumj(HeartIonicSolver<Mesh, SolverType>::M_localMap),
278 M_vectorInfimumm(HeartIonicSolver<Mesh, SolverType>::M_localMap),
279 M_vectorInfimumaa(HeartIonicSolver<Mesh, SolverType>::M_localMap),
280 M_vectorInfimumai(HeartIonicSolver<Mesh, SolverType>::M_localMap),
281 M_vectorInfimumua(HeartIonicSolver<Mesh, SolverType>::M_localMap),
282 M_vectorInfimumui(HeartIonicSolver<Mesh, SolverType>::M_localMap),
283 M_vectorInfimumxr(HeartIonicSolver<Mesh, SolverType>::M_localMap),
284 M_vectorInfimumxs(HeartIonicSolver<Mesh, SolverType>::M_localMap),
285 M_vectorInfimumd(HeartIonicSolver<Mesh, SolverType>::M_localMap),
286 M_vectorInfimumf(HeartIonicSolver<Mesh, SolverType>::M_localMap),
287 M_vectorInfimumfc(HeartIonicSolver<Mesh, SolverType>::M_localMap),
288 M_vectorInfimumpu(HeartIonicSolver<Mesh, SolverType>::M_localMap),
289 M_vectorInfimumpv(HeartIonicSolver<Mesh, SolverType>::M_localMap),
290 M_vectorInfimumpw(HeartIonicSolver<Mesh, SolverType>::M_localMap),
291 M_vectorFn(HeartIonicSolver<Mesh, SolverType>::M_localMap),
292 M_vectorIonicChange(HeartIonicSolver<Mesh, SolverType>::M_localMap),
293 M_solutionGatingH( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
294 M_solutionGatingJ( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
295 M_solutionGatingM( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
296 M_solutionGatingAA( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
297 M_solutionGatingAI( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
298 M_solutionGatingUA( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
299 M_solutionGatingUI( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
300 M_solutionGatingXR( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
301 M_solutionGatingXS( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
302 M_solutionGatingD( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
303     M_solutionGatingF( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
304 M_solutionGatingFCa( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
305 M_solutionGatingPU( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
306 M_solutionGatingPV( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
307 M_solutionGatingPW( HeartIonicSolver<Mesh, SolverType>::M_localMap ),

308 // Concentration
309 M_vectorConcentrationNa( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
310 M_vectorConcentrationK( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
311     M_vectorConcentrationCa( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
312 M_vectorConcentrationCaRel( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
313 M_vectorConcentrationCaUp( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
314 M_vectorIonicChangeCa( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
315 M_vectorIonicChangeNa( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
316 M_vectorIonicChangeK( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
317

```

```

318     M_vectorIonicChangeCaUp( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
319     M_vectorIonicChangeCaRel( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
320
321     //Currents
322     M_vectorCurrentIrel( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
323     M_ionicCurrent( HeartIonicSolver<Mesh, SolverType>::M_localMap ),
324     M_ionicCurrentRepeated( M_ionicCurrent, Repeated ),
325     M_elemVecIonicCurrent ( HeartIonicSolver<Mesh, SolverType>::M_uFESpace.fe() .
326         nbFEDof(), 1 )
327     {
328 }
329 template<typename Mesh, typename SolverType>
330 void CourtemancheRamirezNattel<Mesh, SolverType>::updateRepeated( )
331 {
332
333     M_ionicCurrentRepeated=M_ionicCurrent;
334 }
335
336
337 template<typename Mesh, typename SolverType>
338 void CourtemancheRamirezNattel<Mesh, SolverType>::updateElementSolution( UInt eleID)
339 {
340     M_elemVecIonicCurrent.zero();
341     UInt ig;
342     for ( UInt iNode = 0 ; iNode < HeartIonicSolver<Mesh, SolverType>::M_uFESpace.fe()
343         .nbFEDof() ; iNode++ )
344     {
345         ig = HeartIonicSolver<Mesh, SolverType>::M_uFESpace.dof().localToGlobalMap(
346             eleID, iNode );
347         M_elemVecIonicCurrent.vec()[ iNode ] = M_ionicCurrentRepeated[ig];
348     }
349
350
351
352 template<typename Mesh, typename SolverType>
353 void CourtemancheRamirezNattel<Mesh, SolverType>::solveIonicModel( const vector_Type&
354     u, const Real timeStep )
355 {
356     LifeChrono chronoionmodelsolve;
357     chronoionmodelsolve.start();
358     HeartIonicSolver<Mesh, SolverType>::M_comm->Barrier();
359
360     for ( Int i = 0 ; i < u.epetraVector().MyLength() ; i++ )
361     {
362         Int ig=u.blockMap().MyGlobalElements()[i];
363         Real u_ig=u[ig];
364         computeODECoefficients(u_ig,ig);
365
366         M_ENa = (M_R * M_temperature / M_F ) * log(M_Na0 / M_vectorConcentrationNa[ig]);
367         // fast Na current
368         M_INa = M_gNa * M_solutionGatingM[ig] * M_solutionGatingM[ig] * M_solutionGatingM[
369             ig]* M_solutionGatingH[ig] * M_solutionGatingJ[ig]*(u_ig-M_ENa);

```

```

368
369 // time independant K current
370 M_EKl = (M_R * M_temperature / M_F ) * log(M_K0/M_vectorConcentrationK[ig]);
371 M_KlInf = 1.0/(1.0+exp(0.07*(u_ig+80.0)));
372 M_IKl=M_gKl*M_KlInf*(u_ig-M_EKl);
373
374 // transient outward K current
375 M_EK = (M_R * M_temperature / M_F ) * log(M_K0/M_vectorConcentrationK[ig]);
376 M_Ito = M_gTo*M_solutionGatingAA[ig]*M_solutionGatingAA[ig]*M_solutionGatingAA[ig]
377 *M_solutionGatingAI[ig]*(u_ig-M_EK);
378
379 // ultra rapid delayed rectifier K current
380 M_gKur= 0.005 + 0.05/(1.0+exp(-(u_ig-15.0)/13.0));
381 M_IKur = M_gKur*M_solutionGatingUA[ig]*M_solutionGatingUA[ig]*M_solutionGatingUA[ig]
382 *M_solutionGatingUI[ig]*(u_ig-M_EK);
383
384 // rapid delayed rectifier outward K current
385 M_IKr = M_gKr*M_solutionGatingXR[ig]*(u_ig-M_EK)/(1.0+exp((u_ig+15)/22.4));
386
387 // slow delayed rectifier outward K current
388 M_IKs = M_gKs*M_solutionGatingXS[ig]*M_solutionGatingXS[ig]*(u_ig-M_EK);
389
390 // L-type Ca current
391 M_ICal = M_gCal*M_solutionGatingD[ig]*M_solutionGatingF[ig]*M_solutionGatingFCa[ig]
392 *(u_ig-65.0);
393
394 // Na-K pump
395 M_INaK = M_INaKmax*(1.0/(1.0+0.1245
396 *exp(-0.1*u_ig* M_F/(M_R * M_temperature) )
397 +0.0365/7.0*(exp(M_Na0/67.3)-1.0)
398 *exp(-u_ig* M_F/(M_R * M_temperature) )))
399 *(1.0/(1.0+pow(M_KmNa/M_vectorConcentrationNa[ig],1.5)))*(M_K0/(M_K0+M_KmK0));
400
401 // Na-Ca exchanger
402 M_INaCa = M_INaCamax/((pow(M_KmNa,3.0)+pow(M_Na0,3.0))*(M_KmCa+M_Ca0)*(1.0+0.1*
403 exp((0.35-1.0)*u_ig* M_F/(M_R * M_temperature)))
404 *(exp(0.35*u_ig* M_F/(M_R * M_temperature) )*pow(
405 M_vectorConcentrationNa[ig],3.0)*M_Ca0 - exp((0.35-1.0)
406 *u_ig* M_F/(M_R * M_temperature))*pow(M_Na0,3.0)*
407 M_vectorConcentrationCa[ig]));
408
409 // background current
410 M_ECan=0.5*(M_R * M_temperature)/M_F*log(M_Ca0/M_vectorConcentrationCa[ig]);
411 M_ICab = M_gbCa*(u_ig-M_ECan);
412 //~ M_ENa=(M_R * M_temperature)*log(M_Na0/M_vectorConcentrationNa[ig]); //md
413 04.2012
414 M_INab = M_gbNa*(u_ig-M_ENa);
415
416 // Ca pump
417 M_IpCa = M_IpCamax*(M_vectorConcentrationCa[ig])/(0.0005 +
418 M_vectorConcentrationCa[ig]);

```

```

414 // Sarcoplasmic currents
415
416 // Ca release current from JSR
417 M_vectorFn.epetraVector().ReplaceGlobalValue(ig,0,
418                                     M_Vrel * M_vectorCurrentIrel[ig] -
419                                     ((5.0 * pow(10.0, -1.) / M_F) *
420                                     (0.5 * M_ICal) - 0.2 * M_INaCa));
421                                     // modif 10^2 -> 10^-1 ; md
422                                     04.2012
423 M_vectorCurrentIrel.epetraVector().ReplaceGlobalValue(ig,0,
424                                     M_krel*M_solutionGatingPU[ig]*M_solutionGatingPU[ig]*
425                                     M_solutionGatingPV[ig]*M_solutionGatingPW[ig]*
426                                     M_vectorConcentrationCaRel[ig]-
427                                     M_vectorConcentrationCa[ig]);
428
429 // transfert current from NSR to JSR
430 M_Itr =(M_vectorConcentrationCaUp[ig] -M_vectorConcentrationCaRel[ig])/180.0;
431
432 // Ca uptake current by the NSR
433 M_Iup = M_Iupmax*M_vectorConcentrationCa[ig]/(M_vectorConcentrationCa[ig]+M_Kup);
434                                     //modif md 04.2012
435
436 // Ca leak current by the NSR
437 M_Iupleak = M_Iupmax*M_vectorConcentrationCaUp[ig]/M_consCaUpMax;
438
439
440 ///////////////////////////////////////////////////////////////////
441
442 // The original model from the CRN paper contains buffer modeling. They were
443 // not
444 // implemented here because they have no incidence on transmembrane current.
445 ///////////////////////////////////////////////////////////////////
446
447 M_INatot = 3.0 * M_INaK + 3.0 * M_INaCa + M_INab + M_INa;
448 M_IKtot = -2.0 * M_INaK + M_IK1 + M_Ito + M_IKur + M_IKr + M_IKs;
449 M_ICatot = -2.0 * M_INaCa + M_IpCa + M_ICal + M_ICab;
450
451 // for homogeneity reasons the transmembrane currents IKtot, INatot and ICatot
452 // are multiplied by M_C (this multiplication doesn't appear in the original
453 // paper), but the sarcoplasmic currents are not multiplied
454
455 M_vectorIonicChangeK.epetraVector().ReplaceGlobalValue(ig,0,-(M_IKtot*M_C) /
456                                     (M_F*M_Vi)); // modif : /C md 04.2012
457 M_vectorIonicChangeNa.epetraVector().ReplaceGlobalValue(ig,0,-(M_C*M_INatot) /
458                                     (M_F*M_Vi)); // modif : /C md 04.2012
459 M_vectorIonicChangeCa.epetraVector().ReplaceGlobalValue(ig,0,-(M_C*M_ICatot) /
460                                     (2.0 * M_F * M_Vi)+(M_Vup * (M_Iupleak - M_Iup)+ M_Irel * M_Vrel) / M_Vi)
461                                     / (1.0 + (M_consTrpnMax * M_KmTrpn / (pow( M_vectorConcentrationCa[ig] +

```

```

M_KmTrpn, 2.0))) + (M_consCcmdnMax * M_KmCcmdn / (pow(M_vectorConcentrationCa
[ig] + M_KmCcmdn, 2.0)))); // modif : /C md 04.2012
448 M_vectorIonicChangeCaUp.epetraVector().ReplaceGlobalValue(ig,0,M_Iup-M_Iupleak-
    M_Itr*M_Vrel/M_Vup);
449 M_vectorIonicChangeCaRel.epetraVector().ReplaceGlobalValue(ig,0,(M_Itr-M_Irel)
    /(1.0+M_consCsqnMax*M_KmCsqn/pow(M_vectorConcentrationCaRel[ig]+M_KmCsqn
    ,2.0)));
450 M_ionicCurrent.epetraVector().ReplaceGlobalValue(ig,0,M_INatot + M_IKtot +
    M_ICatot);

451
452 //evolution of gating variables.
453 M_vectorExponentialh.epetraVector().ReplaceGlobalValue(ig,0, exp(-timeStep /
    M_tauh));
454 M_vectorExponentialj.epetraVector().ReplaceGlobalValue(ig,0, exp(-timeStep /
    M_tauj));
455 M_vectorExponentialm.epetraVector().ReplaceGlobalValue(ig,0, exp(-timeStep /
    M_tauf));
456 M_vectorExponentialaa.epetraVector().ReplaceGlobalValue(ig,0, exp(-timeStep /
    M_tauaa));
457 M_vectorExponentialai.epetraVector().ReplaceGlobalValue(ig,0, exp(-timeStep /
    M_tauai));
458 M_vectorExponentialua.epetraVector().ReplaceGlobalValue(ig,0, exp(-timeStep /
    M_tauua));
459 M_vectorExponentialui.epetraVector().ReplaceGlobalValue(ig,0, exp(-timeStep /
    M_tauui));
460 M_vectorExponentialxr.epetraVector().ReplaceGlobalValue(ig, 0, exp(-timeStep /
    M_tauxr));
461 M_vectorExponentialxs.epetraVector().ReplaceGlobalValue(ig, 0, exp(-timeStep /
    M_tauxs));
462 M_vectorExponentiald.epetraVector().ReplaceGlobalValue(ig, 0, exp(-timeStep /
    M_taud));
463 M_vectorExponentialf.epetraVector().ReplaceGlobalValue(ig, 0, exp(-timeStep /
    M_tauf));
464 M_vectorExponentialfc.epetraVector().ReplaceGlobalValue(ig, 0, exp(-timeStep /
    M_taufc));
465 M_vectorExponentialpu.epetraVector().ReplaceGlobalValue(ig, 0, exp(-timeStep /
    M_taupu));
466 M_vectorExponentialpv.epetraVector().ReplaceGlobalValue(ig, 0, exp(-timeStep /
    M_taupv));
467 M_vectorExponentialpw.epetraVector().ReplaceGlobalValue(ig, 0, exp(-timeStep /
    M_taupw));
468 M_vectorInfimumh.epetraVector().ReplaceGlobalValue(ig, 0, M_hinf);
469 M_vectorInfimumj.epetraVector().ReplaceGlobalValue(ig, 0, M_jinf);
470 M_vectorInfimumm.epetraVector().ReplaceGlobalValue(ig, 0, M_minf);
471 M_vectorInfimumaa.epetraVector().ReplaceGlobalValue(ig, 0, M_aainf);
472 M_vectorInfimumai.epetraVector().ReplaceGlobalValue(ig, 0, M_aiinf);
473 M_vectorInfimumua.epetraVector().ReplaceGlobalValue(ig, 0, M_uainf);
474 M_vectorInfimumui.epetraVector().ReplaceGlobalValue(ig, 0, M_uiinf);
475 M_vectorInfimumxr.epetraVector().ReplaceGlobalValue(ig, 0, M_xrinf);
476 M_vectorInfimumxs.epetraVector().ReplaceGlobalValue(ig, 0, M_xsinf);
477 M_vectorInfimumd.epetraVector().ReplaceGlobalValue(ig, 0, M_dinf);
478 M_vectorInfimumf.epetraVector().ReplaceGlobalValue(ig, 0, M_finf);
479 M_vectorInfimumfc.epetraVector().ReplaceGlobalValue(ig, 0, M_fCainf);
480 M_vectorInfimumpu.epetraVector().ReplaceGlobalValue(ig,0, M_puinf);
481 M_vectorInfimumpv.epetraVector().ReplaceGlobalValue(ig, 0, M_pvinf);

```

```

482     M_vectorInfimumpw.epetraVector().ReplaceGlobalValue(ig, 0, M_pwinf);
483 }
484
485 M_vectorExponentialh.globalAssemble();
486 M_vectorExponentialj.globalAssemble();
487 M_vectorExponentialm.globalAssemble();
488 M_vectorExponentialaa.globalAssemble();
489 M_vectorExponentialai.globalAssemble();
490 M_vectorExponentialua.globalAssemble();
491 M_vectorExponentialui.globalAssemble();
492 M_vectorExponentialxr.globalAssemble();
493 M_vectorExponentialxs.globalAssemble();
494 M_vectorExponentiald.globalAssemble();
495 M_vectorExponentialf.globalAssemble();
496 M_vectorExponentialfca.globalAssemble();
497 M_vectorExponentialpu.globalAssemble();
498 M_vectorExponentialpv.globalAssemble();
499 M_vectorExponentialpw.globalAssemble();
500
501 M_vectorInfimumh.globalAssemble();
502 M_vectorInfimumj.globalAssemble();
503 M_vectorInfimumm.globalAssemble();
504 M_vectorInfimumaa.globalAssemble();
505 M_vectorInfimumai.globalAssemble();
506 M_vectorInfimumua.globalAssemble();
507 M_vectorInfimumui.globalAssemble();
508 M_vectorInfimumxr.globalAssemble();
509 M_vectorInfimumxs.globalAssemble();
510 M_vectorInfimumd.globalAssemble();
511 M_vectorInfimumf.globalAssemble();
512 M_vectorInfimumfca.globalAssemble();
513 M_vectorInfimumpu.globalAssemble();
514 M_vectorInfimumpv.globalAssemble();
515 M_vectorInfimumpw.globalAssemble();
516
517 M_vectorFn.globalAssemble();
518 M_vectorCurrentIrel.globalAssemble();
519 M_vectorIonicChangeK.globalAssemble();
520 M_vectorIonicChangeNa.globalAssemble();
521 M_vectorIonicChangeCa.globalAssemble();
522 M_vectorIonicChangeCaUp.globalAssemble();
523 M_vectorIonicChangeCaRel.globalAssemble();
524 M_ionicCurrent.globalAssemble();
525
526
527 M_solutionGatingH-=M_vectorInfimumh;
528 M_solutionGatingH.epetraVector().Multiply(1.,
529                                         M_solutionGatingH.epetraVector(),
530                                         M_vectorExponentialh.epetraVector(),
531                                         0.);
532 M_solutionGatingH+=M_vectorInfimumh;
533 M_solutionGatingJ-=M_vectorInfimumj;
534 M_solutionGatingJ.epetraVector().Multiply(1.,
535                                         M_solutionGatingJ.epetraVector(),
536                                         M_vectorExponentialj.epetraVector(),

```

```

537          0.);
538      M_solutionGatingJ+=M_vectorInfimumj;
539      M_solutionGatingM-=M_vectorInfimumm;
540      M_solutionGatingM.epetraVector().Multiply(1.,
541                                         M_solutionGatingM.epetraVector(),
542                                         M_vectorExponentialm.epetraVector(),
543                                         0.);
544      M_solutionGatingM+=M_vectorInfimumm;
545      M_solutionGatingAA-=M_vectorInfimumaa;
546      M_solutionGatingAA.epetraVector().Multiply(1.,
547                                         M_solutionGatingAA.epetraVector(),
548                                         M_vectorExponentialaa.epetraVector(),
549                                         0.);
550      M_solutionGatingAA+=M_vectorInfimumaa;
551      M_solutionGatingAI-=M_vectorInfimumai;
552      M_solutionGatingAI.epetraVector().Multiply(1.,
553                                         M_solutionGatingAI.epetraVector(),
554                                         M_vectorExponentialai.epetraVector(),
555                                         0.);
556      M_solutionGatingAI+=M_vectorInfimumai;
557      M_solutionGatingUA-=M_vectorInfimumua;
558      M_solutionGatingUA.epetraVector().Multiply(1.,
559                                         M_solutionGatingUA.epetraVector(),
560                                         M_vectorExponentialua.epetraVector(),
561                                         0.);
562      M_solutionGatingUA+=M_vectorInfimumua;
563      M_solutionGatingUI-=M_vectorInfimumui;
564      M_solutionGatingUI.epetraVector().Multiply(1.,
565                                         M_solutionGatingUI.epetraVector(),
566                                         M_vectorExponentialui.epetraVector(),
567                                         0.);
568      M_solutionGatingUI+=M_vectorInfimumui;
569      M_solutionGatingXR-=M_vectorInfimumxr;
570      M_solutionGatingXR.epetraVector().Multiply(1.,
571                                         M_solutionGatingXR.epetraVector(),
572                                         M_vectorExponentialxr.epetraVector(),
573                                         0.);
574      M_solutionGatingXR+=M_vectorInfimumxr;
575      M_solutionGatingXS-=M_vectorInfimumxs;
576      M_solutionGatingXS.epetraVector().Multiply(1.,
577                                         M_solutionGatingXS.epetraVector(),
578                                         M_vectorExponentialxs.epetraVector(),
579                                         0.);
580      M_solutionGatingXS+=M_vectorInfimumxs;
581      M_solutionGatingD-=M_vectorInfimumd;
582      M_solutionGatingD.epetraVector().Multiply(1.,
583                                         M_solutionGatingD.epetraVector(),
584                                         M_vectorExponentiald.epetraVector(),
585                                         0.);
586      M_solutionGatingD+=M_vectorInfimumd;
587      M_solutionGatingF-=M_vectorInfimumf;
588      M_solutionGatingF.epetraVector().Multiply(1.,
589                                         M_solutionGatingF.epetraVector(),
590                                         M_vectorExponentialf.epetraVector(),
591                                         0.);

```

```

592 M_solutionGatingF+=M_vectorInfimumf;
593 M_solutionGatingFCa-=M_vectorInfimumfca;
594 M_solutionGatingFCa.epetraVector().Multiply(1.,
595                                     M_solutionGatingFCa.epetraVector(),
596                                     M_vectorExponentialfca.epetraVector(),
597                                     0.);
598 M_solutionGatingFCa+=M_vectorInfimumfca;
599 M_solutionGatingPU-=M_vectorInfimumpu;
600 M_solutionGatingPU.epetraVector().Multiply(1.,
601                                     M_solutionGatingPU.epetraVector(),
602                                     M_vectorExponentialpu.epetraVector(),
603                                     0.);
604 M_solutionGatingPU+=M_vectorInfimumpu;
605 M_solutionGatingPV-=M_vectorInfimumpv;
606 M_solutionGatingPV.epetraVector().Multiply(1.,
607                                     M_solutionGatingPV.epetraVector(),
608                                     M_vectorExponentialpv.epetraVector(),
609                                     0.);
610 M_solutionGatingPV+=M_vectorInfimumpv;
611 M_solutionGatingPW-=M_vectorInfimumpw;
612 M_solutionGatingPW.epetraVector().Multiply(1.,
613                                     M_solutionGatingPW.epetraVector(),
614                                     M_vectorExponentialpw.epetraVector(),
615                                     0.);
616 M_solutionGatingPW+=M_vectorInfimumpw;
617 M_vectorConcentrationNa+=timeStep*M_vectorIonicChangeNa;
618 M_vectorConcentrationK+=timeStep*M_vectorIonicChangeK;
619 M_vectorConcentrationCa+=timeStep*M_vectorIonicChangeCa;
620 M_vectorConcentrationCaUp+=timeStep*M_vectorIonicChangeCaUp;
621 M_vectorConcentrationCaRel+=timeStep*M_vectorIonicChangeCaRel;
622
623
624 M_solutionGatingH.globalAssemble();
625 M_solutionGatingJ.globalAssemble();
626 M_solutionGatingM.globalAssemble();
627 M_solutionGatingAA.globalAssemble();
628 M_solutionGatingAI.globalAssemble();
629 M_solutionGatingUA.globalAssemble();
630 M_solutionGatingUI.globalAssemble();
631 M_solutionGatingXR.globalAssemble();
632 M_solutionGatingXS.globalAssemble();
633 M_solutionGatingD.globalAssemble();
634 M_solutionGatingF.globalAssemble();
635 M_solutionGatingFCa.globalAssemble();
636 M_solutionGatingPU.globalAssemble();
637 M_solutionGatingPV.globalAssemble();
638 M_solutionGatingPW.globalAssemble();
639 M_vectorConcentrationNa.globalAssemble();
640 M_vectorConcentrationK.globalAssemble();
641 M_vectorConcentrationCa.globalAssemble();
642 M_vectorConcentrationCaUp.globalAssemble();
643 M_vectorConcentrationCaRel.globalAssemble();
644
645 HeartIonicSolver<Mesh, SolverType>::M_comm->Barrier();
646

```

```

647     chronoionmodelsolve.stop();
648     if (HeartIonicSolver<Mesh, SolverType>::M_comm->MyPID()==0)
649         std::cout << "Total ionmodelsolve time " << chronoionmodelsolve.diff() << " s."
650         << std::endl;
651
652
653
654 template<typename Mesh, typename SolverType>
655 void CourtemancheRamirezNattel<Mesh, SolverType>::computeODECoefficients( const Real&
656 u_ig, const Int& ig )//index of the current point in the mapping
657 {
658
659     // Na current
660     if (u_ig >= -40.)
661     {
662         M_ah=0.;
663         M_bh = 1. / (0.13 * (1. + exp( (u_ig + 10.66) / (-11.1) ))));
664         M_aj=0.;
665         M_bj = 0.3 * exp(-2.535e-7 * u_ig) / (1. + exp(-0.1 * (u_ig + 32.)));
666     }
667     else
668     {
669         M_ah = 0.135 * exp((80. + u_ig) / -6.8);
670         M_bh = 3.56 * exp(0.079 *u_ig) + 3.1e5 * exp(0.35 * u_ig);
671         M_aj = (-1.2714e5 * exp(0.2444 * u_ig)-3.474e-5 * exp(-0.04391 * u_ig))*  

672             (u_ig + 37.78) / (1 + exp(0.311 * (u_ig + 79.23)));
673         M_bj = 0.1212 * exp(-0.01052 * u_ig) / (1. + exp(-0.1378 * (u_ig + 40.14)));
674     }
675     M_am = 0.32 * (u_ig + 47.13) / (1. - exp(-0.1 * (u_ig + 47.13)));
676     M_bm = 0.08*exp(-u_ig/11.);
677
678     M_hinf = M_ah / (M_ah + M_bh);
679     M_tauh = 1. / (M_ah + M_bh);
680     M_jinf = M_aj / (M_aj + M_bj);
681     M_tauj = 1. / (M_aj + M_bj);
682     M_minf = M_am / (M_am + M_bm);
683     M_taum = 1. / (M_am + M_bm);
684
685     // transient outward K current
686
687     M_aaa = 0.65*1.0/(exp(-(u_ig+10.0)/8.5)+ exp(-(u_ig-30.0)/59.0));
688     M_baa=0.65*1.0/(2.5+exp((u_ig+82.0)/17.0));
689
690     M_aai = 1.0/(18.53+exp((u_ig+113.7)/10.95));
691     M_bai = 1.0/(35.56+exp(-(u_ig+1.26)/7.44));
692
693     M_tauaa=1.0/((M_aaa+M_baa)*3.0);
694     M_aainf = 1.0/(1.0+exp(-(u_ig+20.47)/17.54));
695
696     M_tauai=1.0/((M_aai+M_bai)*3.0);
697     M_aiinf = 1.0/(1.0+exp((u_ig+43.1)/5.3));
698
699

```

```

700 // ultra rapid delayed rectifier K current
701
702 M_aua=0.65*1.0/(exp(-(u_ig+10.0)/8.5)+exp(-(u_ig-30.0)/59.0));
703 M_bua=0.65*1.0/(2.5+exp((u_ig+82.0)/17.0));
704
705 M_aui = 1.0/(21.0+exp(-(u_ig-185.0)/28.0));
706 M_bui = exp((u_ig-158.0)/16.0);
707
708 M_tauua=(1.0/(M_aua+M_bua))/3.0;
709 M_uainf = 1.0/(1.0+exp(-(u_ig+30.3)/9.6));
710
711 M_tauui=(1.0/(M_aui+M_bui))/3.0;
712 M_uiinf = 1.0/(1.0+exp((u_ig-99.45)/27.48));
713
714
715 // rapid delayed rectifier outward K current
716 M_axr = 0.0003*(u_ig+14.1)/(1.0-exp(-(u_ig+14.1)/5.0));
717 M_bxr = 7.3898e-5*(u_ig-3.3328)/(exp((u_ig-3.3328)/5.1237)-1.0);
718
719 M_tauxr=1.0/(M_axr+M_bxr);
720 M_xrinf = 1.0/(1.0+exp(-(u_ig+14.1)/6.5));
721
722
723 // slow delayed rectifier outward K current
724 M_axs = 4e-5*(u_ig-19.9)/(1.0-exp(-(u_ig-19.9)/17.0));
725 M_bxs = 3.5e-5*(u_ig-19.9)/(exp((u_ig-19.9)/9.0)-1.0);
726
727 M_tauxs=0.5/(M_axs+M_bxs);
728 M_xsinf = pow((1.0+exp(-(u_ig-19.9)/12.7)),(-1.0/2.0));
729
730
731 // L-type Ca current
732
733 M_taud = (1.0-exp(-(u_ig+10.0)/6.24))/(0.035*(u_ig+10.0)*(1.0+exp(-(u_ig+10.0)
    /6.24)));
734 M_dinf=1.0/(1.0+exp(-(u_ig+10.0)/8.0));
735
736 M_tauf=9.0*1.0/(0.0197*exp(-pow(0.0337,2.0)*pow((u_ig+10.0),2.0))+0.02);
737 M_finf=1.0/(1.0+exp((u_ig+28.0)/6.9));
738
739 M_taufca= 2.0;
740 M_fCainf=1.0/(1.0+(M_vectorConcentrationCa[ig]/0.00035));
741
742 // Ca release current from JSR
743 //Fn expression has been simplified, being equivalent as the initial expression (
    // simplification of the power of 10, in the
    //following expressions using Fn). Furthermore, a modification has been done :
        instead of 10^-13, we put 10^-10
744
745 M_taupu=8.0;
746 M_puinf=1.0/(1.0+exp(-(M_vectorFn[ig]*pow(10,4)-3.4175*pow(10,3))/(13.67)));
747
748
749 M_taupv = 1.91 + 2.09 / (1.0 + exp(-(M_vectorFn[ig] * pow(10, 4.0) - 3.4175 * pow
    (10, 3)) / (13.67)));

```

```

750     M_pvinf = 1.0 - 1.0 / (1.0 + exp(-(M_vectorFn[ig] * pow(10, 4.0) - 6.835 * pow(10,
751     2)) / (13.67)));
752
753     M_taupw=6.0*(1.0-exp(-(u_ig-7.9)/5.0))/((1.0+0.3*exp(-(u_ig-7.9)/5.0))*(u_ig-7.9));
754     M_pwinf=1.0-1.0/(1.0 + exp(-(u_ig-40.0)/17.0));
755 }
756
757 template<typename Mesh, typename SolverType>
758 void CourtemancheRamirezNattel<Mesh, SolverType>::computeIonicCurrent( Real
    Capacitance,
759                                         VectorElemental& elvec,
760                                         VectorElemental& /*elvec_u*/,
761                                         FESpace<Mesh, MapEpetra>& uFESpace )
762 {
763     Real Iion_ig;
764     for ( UInt ig = 0; ig < uFESpace.fe().nbQuadPt(); ig++ )
765     {
766         Iion_ig = 0.;
767         for ( UInt i = 0; i < uFESpace.fe().nbFEDof(); i++ )
768             Iion_ig += M_elemVecIonicCurrent( i ) * uFESpace.fe().phi( i, ig );
769         for ( UInt i = 0; i < uFESpace.fe().nbFEDof(); i++ )
770         {
771             elvec( i ) -= Iion_ig * Capacitance *
772                 uFESpace.fe().phi( i, ig ) * uFESpace.fe().weightDet( ig );
773         }
774     }
775 }
776
777
778 //////////////////////////////////////////////////////////////////
779 ////////////////////////////////////////////////////////////////// N.B. The values are taken from the CRN paper and correspond to the
780 // asymptotic values of the model at resting potential. If this method is used,
781 // the corresponding value of u should be resting potential, i.e. -81.2 mV.
782 //////////////////////////////////////////////////////////////////
783
784
785 template<typename Mesh, typename SolverType>
786 void CourtemancheRamirezNattel<Mesh, SolverType>::
787 initialize( )
788 {
789     //initial values of gating variables (asymptotic values of the model at resting
790     //potential)
791     M_solutionGatingH.epetraVector().PutScalar(0.965);
792     M_solutionGatingD.epetraVector().PutScalar(1.37e-4);
793     M_solutionGatingXR.epetraVector().PutScalar(3.29e-5);
794     M_solutionGatingAI.epetraVector().PutScalar(0.999);
795     M_solutionGatingUI.epetraVector().PutScalar(0.999);
796     M_solutionGatingM.epetraVector().PutScalar(2.91e-3);
797     M_solutionGatingJ.epetraVector().PutScalar(0.978);
798     M_solutionGatingF.epetraVector().PutScalar(0.999);
799     M_solutionGatingXS.epetraVector().PutScalar(1.87e-2);
800     M_solutionGatingAA.epetraVector().PutScalar(3.04e-2);
801     M_solutionGatingUA.epetraVector().PutScalar(4.96e-3);

```

```

802 M_solutionGatingFCa.epetraVector().PutScalar(0.775);
803 M_solutionGatingPV.epetraVector().PutScalar(1.0);
804 M_solutionGatingPU.epetraVector().PutScalar(0.0);
805 M_solutionGatingPW.epetraVector().PutScalar(0.999);
806
807
808
809 //initial concentrations.
810 M_vectorConcentrationNa.epetraVector().PutScalar(11.2);
811 M_vectorConcentrationK.epetraVector().PutScalar(139.0);
812 M_vectorConcentrationCa.epetraVector().PutScalar(1.02e-4);
813 M_vectorConcentrationCaUp.epetraVector().PutScalar(1.49);
814 M_vectorConcentrationCaRel.epetraVector().PutScalar(1.49);
815
816 M_solutionGatingH.globalAssemble();
817 M_solutionGatingD.globalAssemble();
818 M_solutionGatingXR.globalAssemble();
819 M_solutionGatingAI.globalAssemble();
820 M_solutionGatingUI.globalAssemble();
821 M_solutionGatingM.globalAssemble();
822 M_solutionGatingJ.globalAssemble();
823 M_solutionGatingF.globalAssemble();
824 M_solutionGatingXS.globalAssemble();
825 M_solutionGatingAA.globalAssemble();
826 M_solutionGatingUA.globalAssemble();
827 M_solutionGatingFCa.globalAssemble(),
828 M_solutionGatingPV.globalAssemble();
829 M_solutionGatingPU.globalAssemble();
830 M_solutionGatingPW.globalAssemble();
831 M_vectorConcentrationNa.globalAssemble();
832 M_vectorConcentrationK.globalAssemble();
833 M_vectorConcentrationCa.globalAssemble();
834 M_vectorConcentrationCaUp.globalAssemble();
835 M_vectorConcentrationCaRel.globalAssemble();
836 }

```