Modelling Ionic Behavior within the Glycocalyx

Bryan Ho

Advisor: Dr. James Sterling

Abstract

The glycocalyx interacts with saline solutions and creates an electric potential gradient that controls the distribution of the ions it interacts with. Understanding this electrostatic behavior may provide further insight into the role the glycocalyx plays in the body, including its ability to absorb drugs and its role in innate immunity against infection. This paper mathematically models the glycocalyx-salt system using Poisson's equation under various conditions, including the effects of the solvent and the presence of different ionic valencies. A transient simulation is also created utilizing Nernst-Planck equations, which can model a changing system over time based on specified initial conditions. Specifically, we mathematically compute a system at steady state and demonstrate its stability within the transient model, creating a distribution profile that allows us to better understand the nature of the electrical potential within the glycocalyx. We find that the glycocalyx creates an electrical Donnan potential in order to move ions across its membrane and to stabilize the flux of ions from the outside environment, establishing an extracellular and mucosal surface barrier. We report details on how net electroneutrality is attained through establishment of the Donnan potential for various salt concentrations and ion-pairing binding constants for both monovalent and divalent cations.

Introduction

The glycocalyx is a network of densely-packed, polysaccharide molecules that lines extracellular and mucosal surfaces, acting as a semipermeable membrane that interacts with the outside environment.[1] The contact between the glycocalyx and the outside salt solution allows ions from the salt to permeate into the glycocalyx. Ions experience electrostatic forces that drive them to become transiently-bound to the glycocalyx sugar-chains. The sugars come in the form of proteoglycans, mucins, and glycosaminoglycans that are negatively charged with carboxylates and sulfates. For the purposes of this paper, we will treat all sugar-bound anions as being of a specific type (e.g. sulfates) with a charge of -1, creating a model with a uniform, negatively-charged sulfate membrane acting as the glycocalyx. The interaction between the diffuse glycan layer, henceforth referred to as the gel, and a salty environment creates a system of free-moving cations and anions (e.g. sodium and chloride ions) that are moved into the gel and come into contact with the gel's sulfates, potentially pairing with them in the case of the cations. The ions will interact in such a way as to promote electroneutrality, with both salt ions moving into the membrane to create as neutral of a layer as possible.

Because the negatively-charged sulfates are tethered to the surface, a negative Donnan potential (with the salt solution far from the interface being ground) is created. The presence of the distributed charges in the glycocalyx ensures an overall unequal distribution of ions. Figure 1 offers a simple illustration of the gel-salt complex. It is taken directly from Sterling and Baker, and here we consider this geometry as well. The gel and the salt layers are distinctly separated, but ions from the salt can be seen moving across the interface and interacting with the bounded sulfates. The variables $c_{j,0}$ and $c_{i,0}$ are, respectively, the initial integral charge-densities of the sugar-bounded sulfates and free salt concentrations far from the interface, with $c_{i,0}$ representing the concentrations of both cations and anions in a salt with 1:1, monovalent ions.

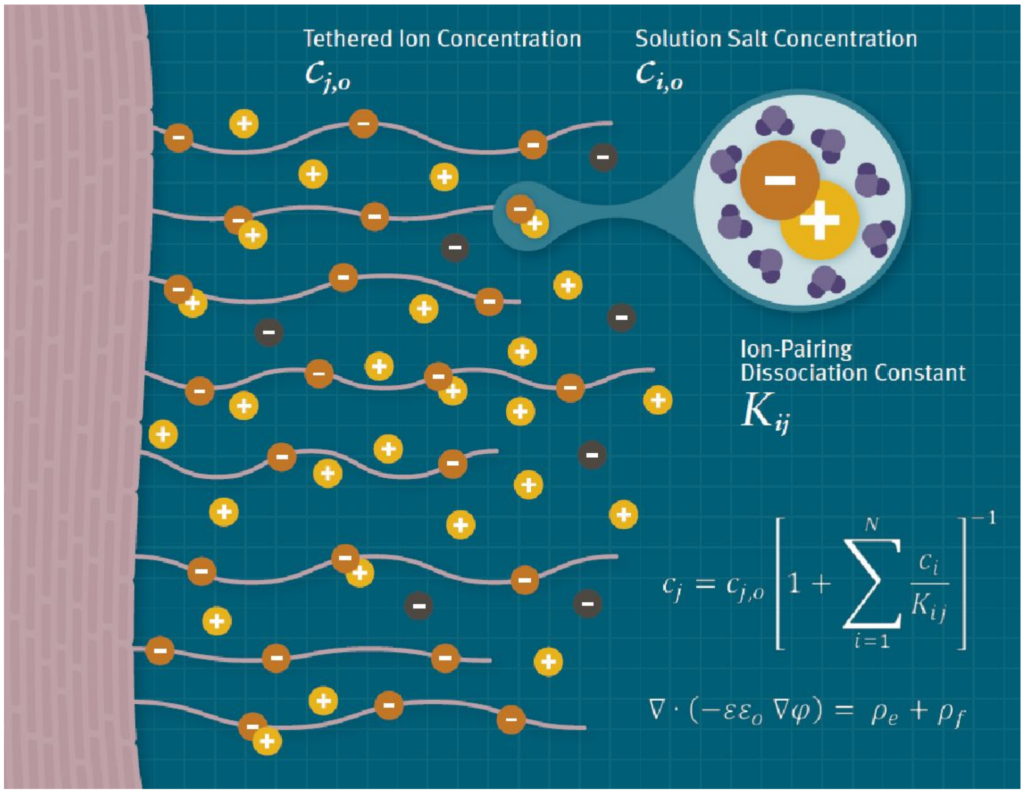
Tethered Ion Concentration
$C_{j,o}$
Solution Salt Concentration
$C_{i,o}$
Ion-Pairing
Dissociation Constant
K_{ij}
$c_j = c_{j,o}\left[1 + \sum_{i=1}^{N} \frac{c_i}{K_{ij}}\right]^{-1}$
$\nabla \cdot (-\varepsilon\varepsilon_o \nabla\varphi) = \rho_e + \rho_f$

Figure 1. Depiction of the gel and salt solution, including cationic-sulfate binding.[1]

Understanding the electrostatic characteristics of the glycocalyx can provide fundamental knowledge about the molecular nature of the mucosal membrane. The glycocalyx exists in all organs in the body, including the mouth, eyes, lungs, and gastrointestinal and reproductive tracts; therefore, creating a model of the distribution of ions and electrical potentials can provide insight into developing drugs that can work most effectively in these environments. Mucosal glycocalyx surfaces have also been observed to be the most prone to infection, as they are direct barriers to incoming flux of material from the outside solution and serve as a crucial defense against invasive bacteria.[2] Even severe genetic diseases, such as cystic fibrosis and Sjogren's syndrome, can disrupt functional glycocalyxes, so a clearer understanding of the glycocalyx as a diffuse semipermeable membrane can be of significant contribution to further research. A literature review indicates that electrostatic models of the glycocalyx do not adequately address the ion pairing (contact ion pairing or solvent-separated ion pairing) that is needed to accurately represent the electrostatics of the glycocalyx.[1] In this paper, we aim to illustrate the mechanisms of both ion screening and ion pairing in a number of models of the gel-salt system by mathematical modeling and computational analysis of ionic and Donnan potential behavior.

Materials and Methods

For this paper, a computational model in Python has been used to simulate the glycocalyx-salt layer system. The gel-salt complex is represented as a one-dimensional system with both the gel and salt layers present and separated on a number line composed of a dimensionless measurement for distance, **x**. The negative domain, with 0 inclusive, represents the gel layer, while the positive domain represents the salt solution, meaning that the line x = 0 represents the gel-salt interface and the end of the glycocalyx. Because of the nature of the system we are trying to model, sulfates will only exist on the negative domain and at 0, for they represent the glycocalyx as fixed charges. There is no sulfate concentration within the positive domain. For this simulation, the domain ranges from -10 to 10, with dx being set to 0.05 to provide good spatial resolution of the differential equations. The code for each model can be found in the appendices. Code that is identical to the steady state code save for specific parameters and equations will not be included to avoid unnecessary repetition.

<u>The Steady State</u>

The first step in understanding the electrokinetics of the system is to model the electrical potential distribution at steady state under given conditions. The model applies Poisson's equation of electrostatics, which brings together elements of ionic movement and binding.[3]

$$\nabla \cdot (-\varepsilon\varepsilon_0 \nabla\varphi) = \rho_e + \rho_f \quad \textbf{(1)}$$

This equation is a second-order gradient evaluation of the electrical potential using the charge density decomposed into free and fixed charges. We nondimensionalize the equation by the Debye length so that the dielectric constant ε and the permittivity of free space ε_0 are absorbed into the x-variable to simplify our model. ρ_e represents the charge density of free moving ions from the salt, while ρ_f represents the charge density of the fixed sulfates. Φ is the value of the electrical potential and it is nondimensionalized by the thermal voltage such that the Donnan potential is written as **y**. Evaluating the second derivative in terms of the gel-salt complex and replacing both ρ values with their respective Boltzmann equilibrium equations, we arrive at

$$\frac{\partial^2 y}{\partial x^2} = c_{i,0}(e^y - e^{-y}) + c_{j,0}(1 + \frac{c_{i,0}}{K}e^{-y})^{-1} \quad \textbf{(2)}$$

where K is the dissociation constant of the cation-sulfate binding interaction. Nondimensionalizing equation (2) by dividing it by $c_{j,0}$ gives the final equation

$$\frac{\partial^2 y}{\partial x^2} = S(e^y - e^{-y}) + (1 + Pe^{-y})^{-1} \quad \textbf{(3)}$$

where S (the screening constant) measures the relative charge densities resulting in electrophoretic movement of cations into the glycocalyx and P (the pairing constant) measures the binding ability of the cations with the sulfates. Here, $S = c_{i,0} / c_{j,0}$ and $P = c_{i,0} / K$. For this model and all of the following models, we set $c_{j,0}$ to be equal to a dimensionless numerical value of 10. We also assume that the reactions involved are instantaneous so that the movement of ions do not create magnetic fields and the above equations apply to the system at every state, not just the steady state. Equation (3) governs the electrical potential within the gel while the equation for the salt layer has $\rho_f = 0$ due to the absence of the sulfates. Therefore, the potential equation for the salt layer is simply

$$\frac{\partial^2 y}{\partial x^2} = S(e^{y} - e^{-y}) \tag{4}$$

With the electrical potential equations for both sides of the interface, combining equations (3) and (4) allows for the numerical computation for the steady state distribution of the ion concentrations and the Donnan potential. Because we divide the system into two domains, four boundary conditions must be calculated: the far left gel potential, the far right salt potential, and both the potential and the gradient of the potential where the layers meet, which must be matched on both sides of the interface. For this first simulation, we use weak screening and pairing such that S = 0.1 and P = 0.1. The far right boundary will be our reference ground, so its value is kept at 0. To find the far left boundary, we assume that the ions at the far left are settled, so (3) is equal to 0. Solving the algebraic equation for the Donnan potential, we find that y_{gel} = -1.839. To find the interface potential, a boundary solver was created that iterates through different values of y between 0 and -1.839 and evaluates both (3) and (4) in order to find the y at which the value and slope of the electrical potential for both equations are equal, while also enforcing previously stated boundaries. For the given system, it was found that y_0 = -1.187. These jump condition values can then be used as boundary conditions in graphing both equations to form the potential distribution. All Python code and calculations for this system can be found in Appendix A.

Transient Ion Distribution

Having found jump conditions and steady state profile solutions, we are interested in understanding how a given gel-salt initial condition will evolve over time to match that distribution. This would provide a transient depiction of the gel-salt complex, and creating a model for it would allow for a check to the validity of the steady state solution. An initial test can be done by setting the transient simulation to be already at the steady state, and tracking how the distribution changes over time. This can be done by creating a simulation that plots all three ions on discrete points on the x domain and continuously updates their concentrations based on ionic behavior, while also computing the electrical potential at every point in the domain. Such calculations involve Poisson Nernst Planck (PNP) equations that model how the ions interact and move over time.

A reevaluation of (1) can be done that incorporates measurements of the concentrations of each ion involved in the system. The Poisson equation is then given by:

$$\frac{\partial^2 \phi}{\partial x^2} = \frac{F}{-\epsilon\epsilon_0}(-C_s + C_c - C_a) \tag{5}$$

where F is Faraday's Constant, and C_s, C_c and C_a represent the concentrations of free, unpaired sulfates, salt cations, and salt anions at that x, respectively. Once again, we set the dielectric constants as well as F to be 1 and nondimensionalize with respect to $c_{j,0}$:

$$\frac{\partial^2 y}{\partial x^2} = \frac{-C_s + C_c - C_a}{C_{j,0}} \tag{6}$$

Equating (3) and (6) allows for the concentration profiles of each ion at the steady state where the value of the concentration at a specific x is a function of the electrical potential at that point:

$$C_s = c_{j,0}(1 + Pe^{-y})^{-1}$$
$$C_c = c_{j,0}Se^{-y} \qquad\qquad C_a = c_{j,0}Se^{y} \tag{7}$$

Utilizing the steady state solution found earlier, values of the steady state electrical potentials can be used to find the concentration of each ion at every point, allowing us to initialize the transient model at steady state.

To model the movement of the ions over time, the Nernst-Planck equation is used.

$$\frac{\partial c_i}{\partial t} = \nabla \cdot \left(\frac{z_i}{|z_i|} u_i c_i \nabla \phi + D_i \nabla c_i \right) + \sum_{j=1}^{N} \left(-k_{aij} c_i c_j + k_{dij} c_{ij} \right) \tag{8}$$

where u_i and D_i are the electrophoretic mobility and diffusivity constants for each individual ion, i. For simplicity, μ and D for all ions will be set to 1. The constants k_a and k_d are the association and dissociation constants of the cation-sulfate reaction, where $K = k_d / k_a$. For this model, we set k_d to be 10 and k_a to be 1. Evaluating (8) for each ion results in

$$\begin{aligned}
\frac{\partial C_s}{\partial t} &= && -k_a C_s C_c + k_d (C_{j,0} - C_s) \\
\frac{\partial C_c}{\partial t} &= +\mu_c C_c \frac{\partial^2 \phi}{\partial x^2} + \mu_c \frac{\partial C_c}{\partial x}\frac{\partial \phi}{\partial x} + D_c \frac{\partial^2 C_c}{\partial x^2} && -k_a C_c C_s + k_d (C_{j,0} - C_s) \\
\frac{\partial C_a}{\partial t} &= -\mu_a C_a \frac{\partial^2 \phi}{\partial x^2} - \mu_a \frac{\partial C_a}{\partial x}\frac{\partial \phi}{\partial x} + D_a \frac{\partial^2 C_a}{\partial x^2} &&
\end{aligned} \tag{9}$$

These equations allow for the calculation of each ion present in the gel layer. Due to the absence of sulfates in the salt layer, a reevaluation of (9) leads to a series of equations for the salt solution:

$$\begin{aligned}
\frac{\partial C_s}{\partial t} &= 0 \\
\frac{\partial C_c}{\partial t} &= +\mu_c C_c \frac{\partial^2 \phi}{\partial x^2} + \mu_c \frac{\partial C_c}{\partial x}\frac{\partial \phi}{\partial x} + D_c \frac{\partial^2 C_c}{\partial x^2} \\
\frac{\partial C_a}{\partial t} &= -\mu_a C_a \frac{\partial^2 \phi}{\partial x^2} - \mu_a \frac{\partial C_a}{\partial x}\frac{\partial \phi}{\partial x} + D_a \frac{\partial^2 C_a}{\partial x^2}
\end{aligned} \tag{10}$$

This series of equations allows us to evaluate the changes in each ion over time and to simulate an evolving model in which each value can be continuously updated over a set period of time via first-order Euler's method calculations. Equation (6) can then be used to to find the electric potential by integrating over its values twice. For this simulation, dt = 0.0005 was used for all calculations involving time derivatives. One confirmation that the computation is accurate is if the steady state solution remains stable. Another test involves perturbing the steady state model and observing if the transient system evolves into the steady state distribution over time. All code and calculations for the transient state can be found in Appendix B.

Solvent-Separated Ion Pairing

The ionic pairing used in the previous models were simplistic representations of the complex ionic kinetics that can occur in such a system. The previous models assumed that the salt cations and sulfates would form contact ion pairs and could only dissociate back into free ions, requiring a single equilibrium constant to characterize ion-pairing. However, ions are well-known to bind with the solvent water molecules prior to binding with other ions. Thus, there are two energy-minima known as CIP and SIP pairing as shown in Figure 2.

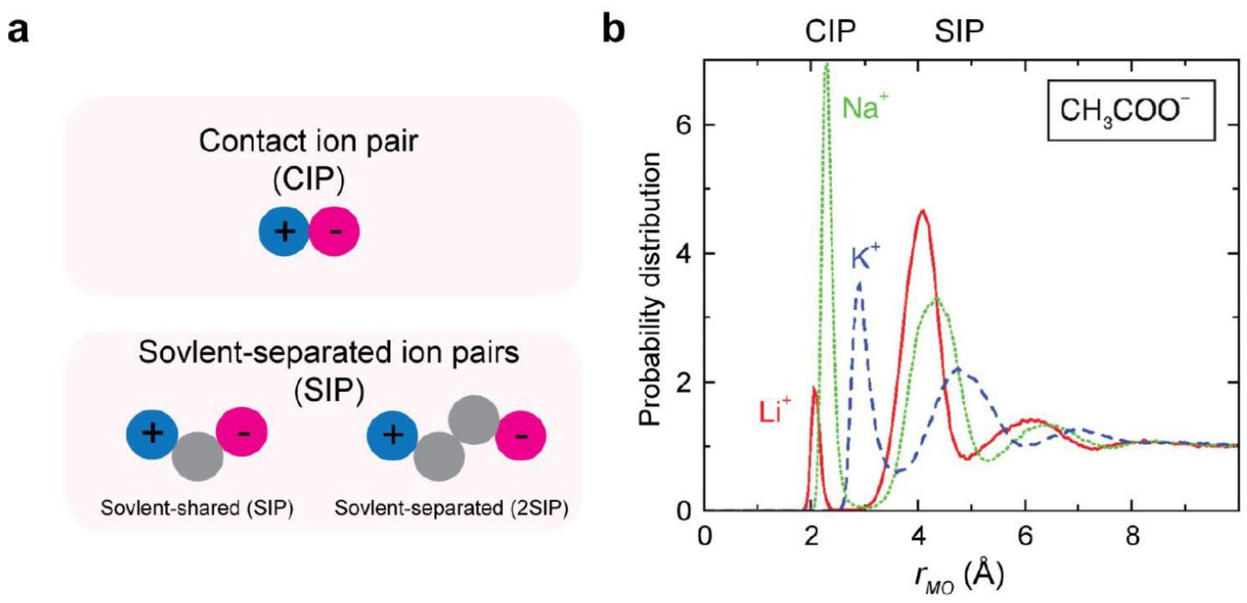

Figure 2. Illustrations of both solvent-separated and contact ion pairing. 2b illustrates the distribution of various pairs based on the distance between the base ions.[4]

As Iwahara and others observed, the solvent can be present within the ionic complex, leading to a solvent-separated ion pair, with the number of solvent molecules varying based on the hydration characteristics of the ions.[4] These solvent-separated ion pairs can then eject the solvent and form into normal, contact ion pairs, as demonstrated by the increased distribution of more closely-paired ions in Figure 2b. The peaks of each ever graph of ionic pairs represents a different degree of solvent separation, as measured by the distance (in Angstroms) between the two base ions. The formation of CIP from SIP is measured using a secondary dissociation constant, similar to K used in the previous models. To promote clarity, this paper will use the inverse of that secondary K and call it Q. Rewriting (3) with this new variable results in

$$\frac{\partial^2 y}{\partial x^2} = S(e^{y} - e^{-y}) + [1 + P(1+Q)e^{-y}]^{-1} \quad \textbf{(11)}$$

A steady state solution using this new model can then be found using the same methods as above. We assume that the ion pairs must first bind as SIP and then have the possibility of further transitioning into CIP. The code and calculations for this alternative pairing model are identical to those in Appendix A, only with substituted parameters and equations. Supplementary code can be found in Appendix C.

Divalent Salt Ion Pairing

The final case that this paper considers is one wherein the salt is not composed of 1:1, monovalent ions. Instead, we consider a system in which the salt has a divalent cation, meaning the charges are in a 2:1 ratio. As an example, such a system could exist if the salt is comprised of magnesium (+2) and chloride (-1) ions. Such a system offers new methods of binding, as monovalent sulfate group and magnesium pairs would still have net +1 positive charge, allowing it to bind with another sulfate group on the glycan. We model this using a secondary pairing constant that represents the reaction constant from primary to secondary cationic binding. Rederiving Poisson's equation (3) in terms of this divalent system leads to:

$$\frac{\partial^2 y}{\partial x^2} = S(e^y - e^{-2y}) + \frac{(-1 - \frac{P_1 c_{i,0} e^{-y}}{2} + [(1 + \frac{P_1 c_{i,0} e^{-y}}{2})^2 + 4P_1 P_2 c_{j,0} c_{i,0} e^{-y})]^{\frac{1}{2}})}{2P_1 P_2 c_{j,0} c_{i,0} e^{-y}} \tag{12}$$

and accordingly, the equation for the salt side is:

$$\frac{\partial^2 y}{\partial x^2} = S(e^y - e^{-2y}) \tag{13}$$

Note the factors of 2 in the equation, which reflect the presence of the divalent cation. For Equation (12), $c_{i,0}$ represents the initial concentration of salt anions, meaning the concentration of the salt cation is half of that value far from the interface. Solving this equation once again leads to a graph of the steady state solution. The code and calculations for this divalent model are identical to those in Appendix A, only with substituted parameters and equations reflecting the divalent nature of the model. Supplementary code can be found in Appendix D.

Results

All of the aforementioned models were programmed using Python code and associated libraries. All of the following graphs were created using this code, which can be found within the appendices.

The Steady State

Solving for the steady state results in Figure 3. Figure 3a shows the computed solution of the potential distribution, which can be seen as a smooth and somewhat symmetrical profile that ranges from 0 to the calculated left boundary potential. Figure 3b provides a graph of the potential, its derivative, and its second derivative. As can be seen, both first and second derivatives become 0 near the boundaries, but fluctuate near the interface, indicating that the variations occur within a few Debye lengths of the interface in a so-called double-double layer around the interface.

To fully understand the effects of the screening and pairing constants, which are the only independent variables that were controlled in this system, a surface plot is created to graph the relationship between S, P, and the far left boundary potential (the Donnan potential), found in Figure 4. From the graph, it can be seen that increases in both S and P provide a decrease in the magnitude of the Donnan potential. Increased screening drives more cations from the salt into the gel, and more pairing allows for more sulfates to bind, decreasing the overall free anionic charge of the glycocalyx.

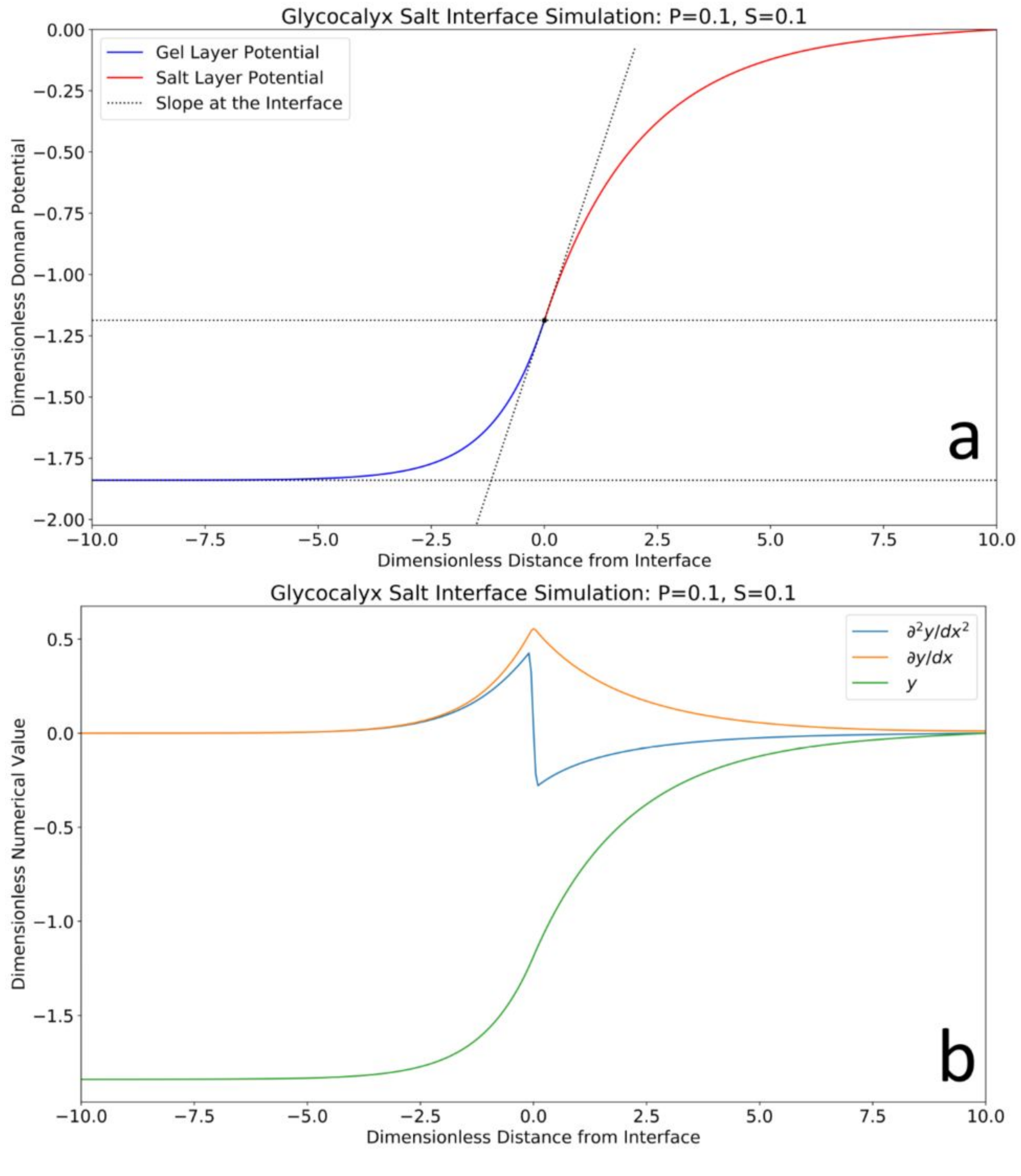

Figure 3. The steady state analytical solution at specified values. Note that Figure 3b also includes curves for the various derivatives of the Donnan potential.

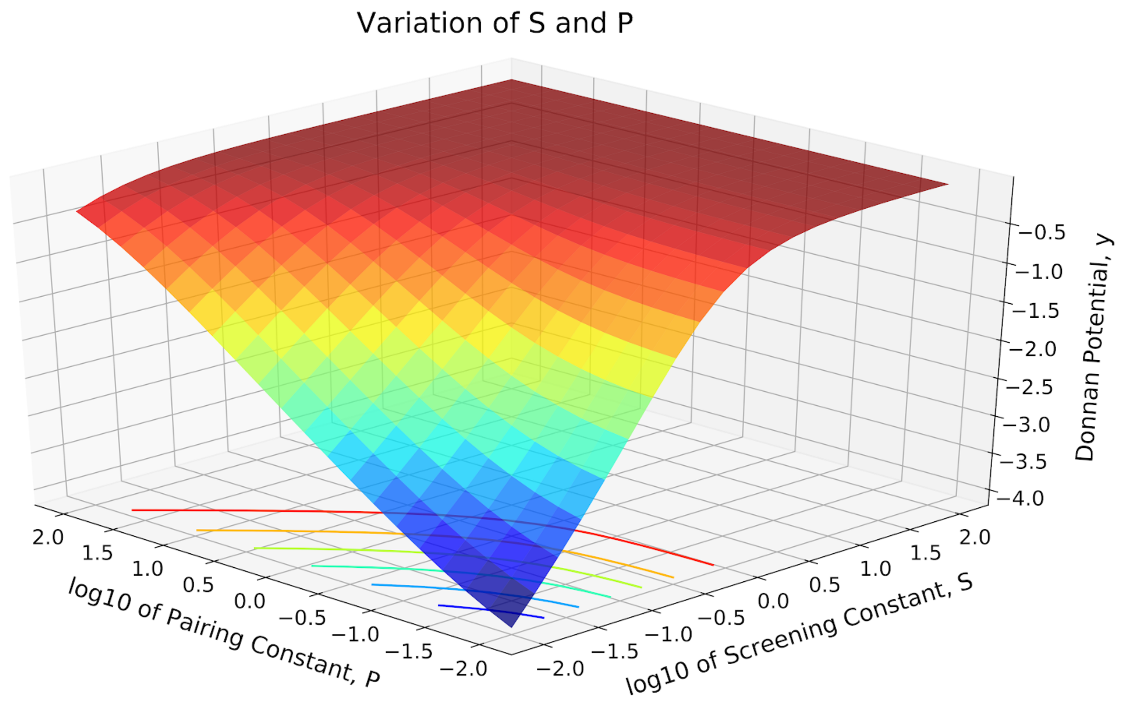

Figure 4. A surface plot illustrating the relationship between the screening and pairing constants with the Donnan potential of the glycocalyx.

Finally, Figure 5 shows the distribution of the concentrations of each ion using the steady state solution and equations from (7). These will be used in the transient model in order to initialize the system at steady state.

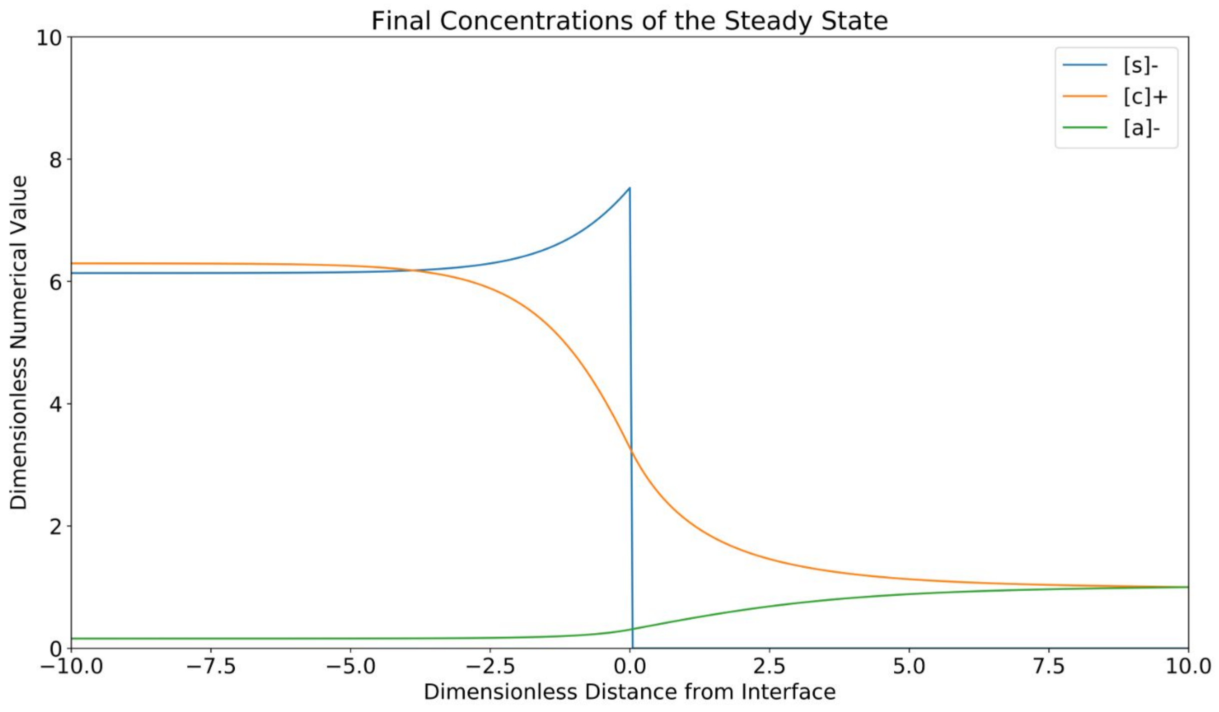

Figure 5. The concentration profiles of each ion involved at the steady state solution. These values are used to initialize the transient model.

Transient Ion Distribution

By initializing a transient model with our previously-computed steady state, we can observe the evolution of the system and confirm that it is indeed a stable steady state. Figure 6a shows a waveform graph depicting the change in the potential distribution over a simulated period of 20 dimensionless time units (recall that *dt* was set to be 0.0005). Figure 6b illustrates the final potential at that time mark. Note that left bound potential at the very first time step is -1.722, a -6.36% error in what our steady state solution predicted. We believe this error was caused by numerical error accumulated in the calculations made within the transient simulation. Nonetheless, the distribution changes slightly over time, as the waveform indicates that the curve slowly moves upward. The overall change per timestep, however, steadily decreases over time, indicating that the system itself is somewhat stable, with only the error causing slight deviations in measured values. We conclude that the transient model does converge to values close to the steady state solution. The profiles for the final potential, as seen in Figure 6b, is also of relatively the same shape and values of the steady-solution. Figure 7 is the concentration profiles of the ions at the final timestep, further demonstrating a similarity to the steady state model. We believe that our usage of a first-order numerical method results in error that could be addressed using higher-order numerical methods.

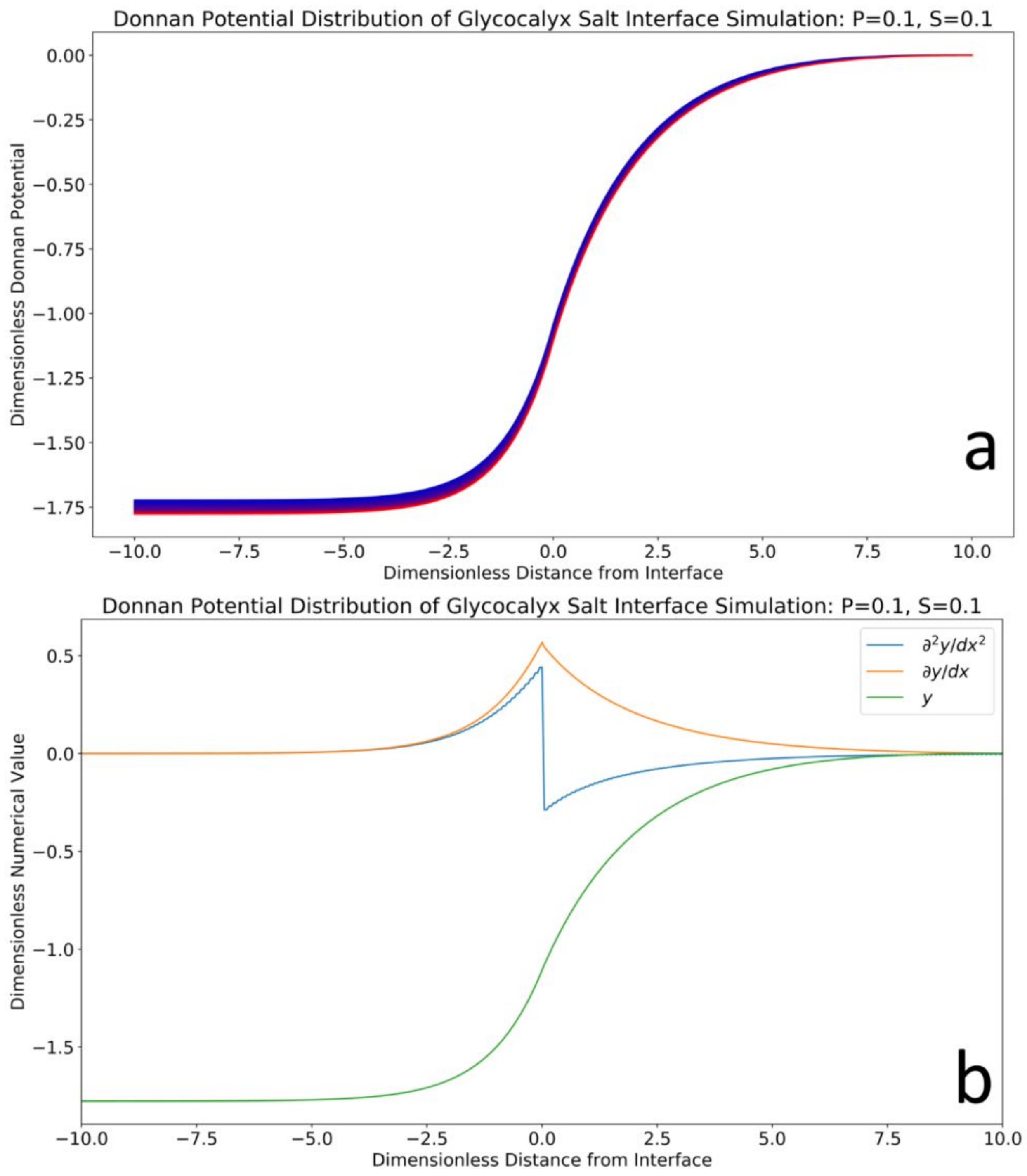

Figure 6. Graphs of the Donnan potential distribution in the evolving transient system, initialized at the steady state. For 6a, the graph evolves from blue to red, representing the continuous change from time 0 to 20 time units. 6b is a profile of the final distribution at time 20.

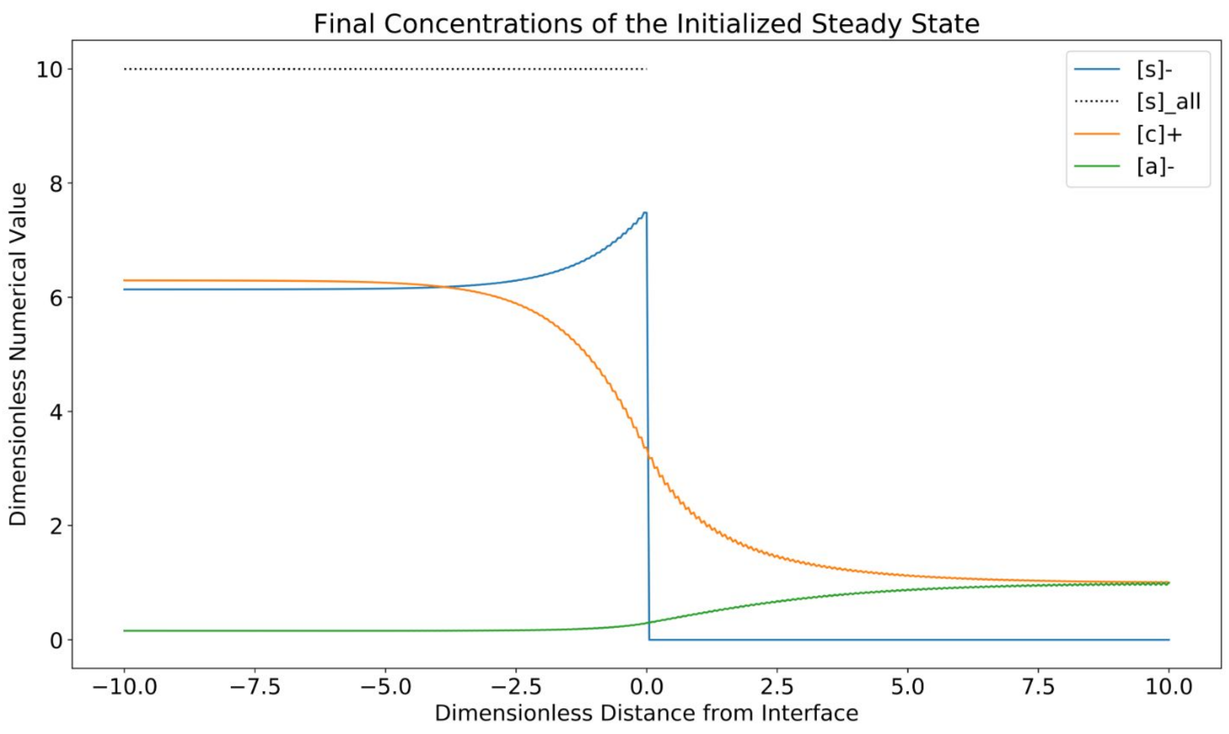

Figure 7. The concentration profiles of each ion in the transient system initialized at steady state at time 20.

A second case for the transient case involves initializing the system at a perturbed state and observing if the system eventually comes to the same steady-solution as before. This is accomplished by taking the initial steady state concentrations and reflecting both the salt cation and anion values across the interface, thereby keeping the same integral charge volume. Figure 8a, which is this transient system run from time 0 to 30, demonstrates that the system returns to a steady state solution, with the final left potential at that time step being -1.848, essentially equivalent to that of the steady state system. Figure 9 depicts the final concentrations of this model, which also appears to be identical to the steady state profiles. The code and calculations for this system are identical to that of the initialized steady state model, with only the initial concentrations changed.

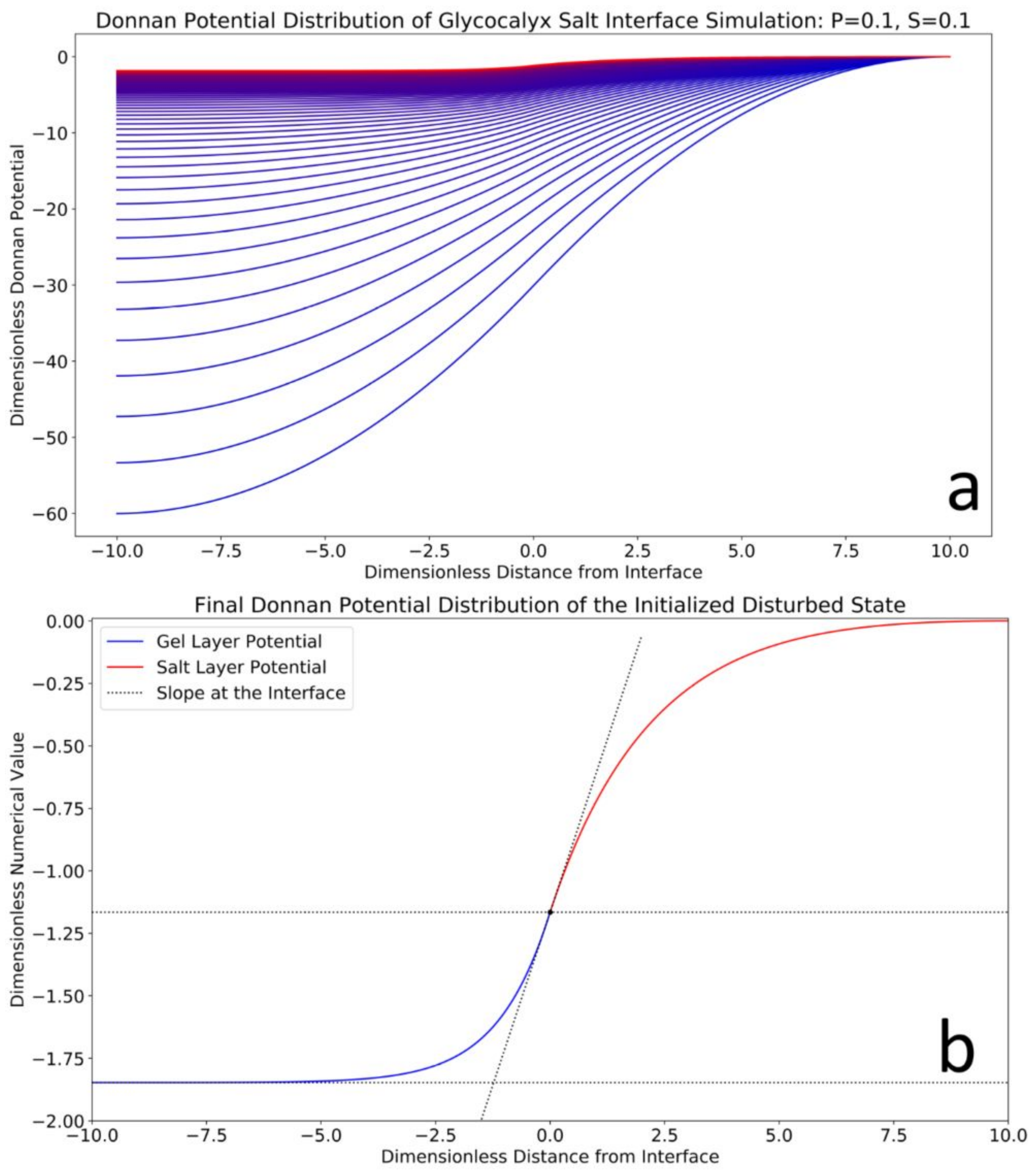

Figure 8. Graphs of the Donnan potential distribution in the evolving transient system initialized at a disturbed state. For 6a, the graph evolves from blue to red, representing the continuous change from time 0 to 30. 6b is a profile of the final distribution at time 30.

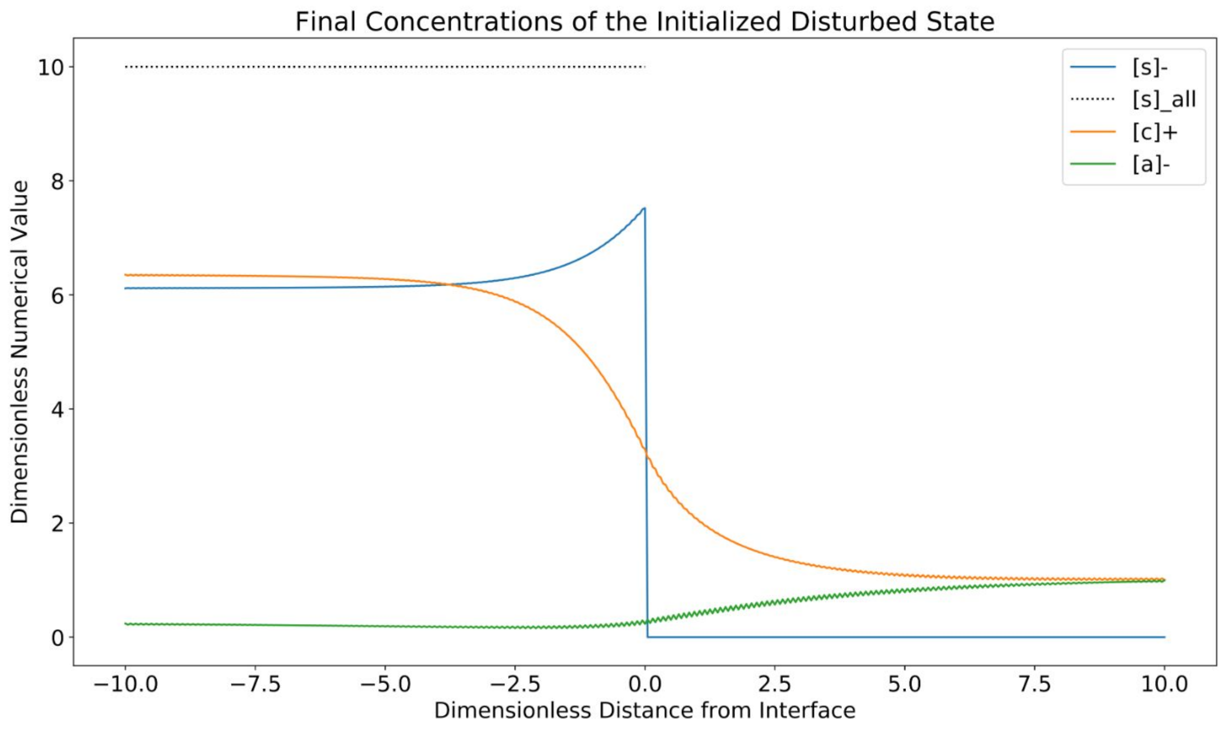

Figure 9. The concentration profiles of each ion in the initialized disturbed state system at time 30.

Solvent-Separated Ion Pairing

We conduct the same analysis for a model that explicitly considers equilibria of both SIP and CIP, known as the Eigen-Tamm model. At the same S and P values of 0.1, with Q = 0.1, we see that the potential distribution is almost identical to that of the steady state solution listed before, with this model's distribution shown in Figure 10a. The only slight variation in curvature is due to the left-bound potential being equal to -1.813, only a 1.14% deviation from the previous steady state. As Figure 10b demonstrates, however, different variations of P and Q will greatly alter the jump conditions of the left-boundary potential. Due to Q being the inverse of the dissociation constant for SIP ion pairs, increases in Q reflect a decrease in the dissociation of SIP into CIP. Consequently, this results in an increased amount of total ion pairs and a kind of neutralization of the electrical potential.

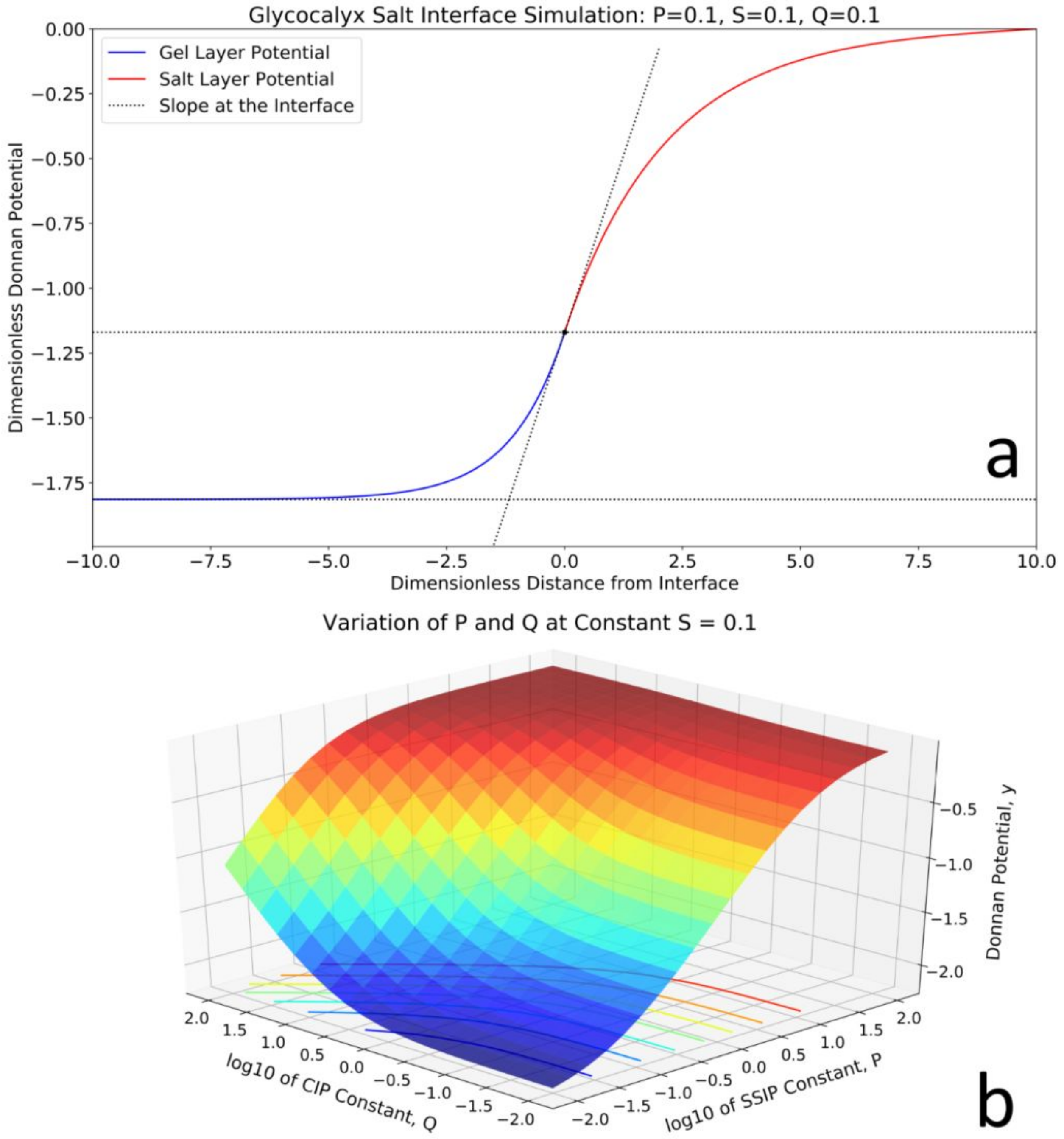

Figure 10. The steady state solution and P-Q surface plot of the solvent-separated system.

Divalent Salt Ion Pairing

The final case is another steady state solution. As can be clearly seen in Figure 11a, the presence of the divalent ion substantially changes the potential distribution. Given S = 0.1, P1 = 0.1, and P2 = 0.1, the left-bound potential only goes to -1.165, a 36.6% deviation from the monovalent system. Figure 11b illustrates the significant effects that the secondary reaction creates, as the addition of an additional charge greatly increases the potential for more ionic bonding with the same concentration of sulfates. It does appear that increases in both P1 and P2 lead to a decrease in the magnitude in the potential, with it being noteworthy that the effects of P2 seem dependent on P1. Increases in bonding, as discussed before, are expected to decrease the potential, and it is only logical that the secondary reaction also drives ion pairing.

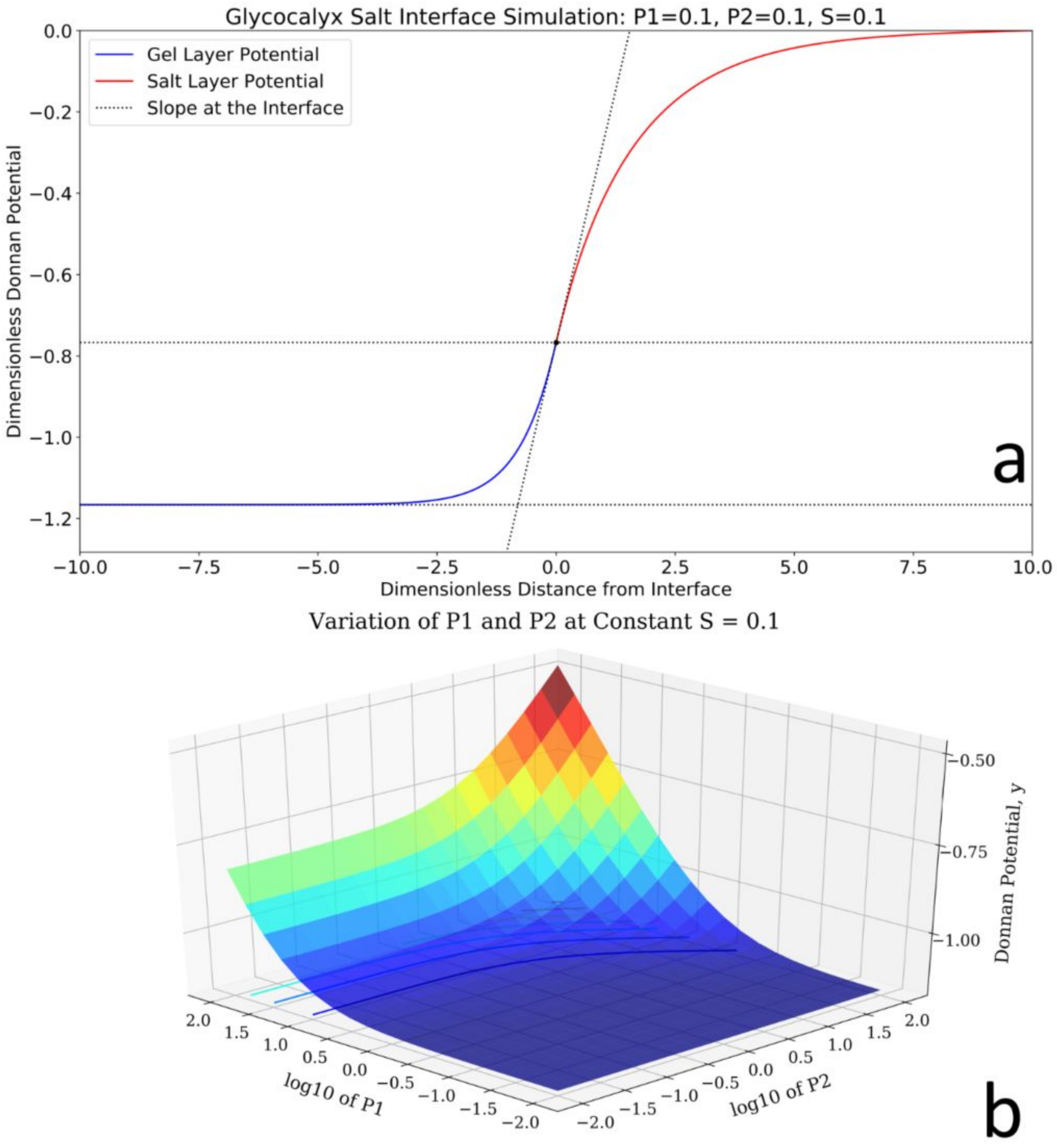

Figure 11. The steady state solution and P1-P2 surface plot of the divalent salt system.

Discussion

The glycocalyx models presented in this paper provide details regarding and insight into the electrostatic properties and ion charge distributions of the glycocalyx membrane. The models are used to compute profiles of the potential from ground potential 0, assumed for the saline solution far from the interface, to a specified Donnan potential at the gel boundary, also far from the interface. The relationship between the salt cations and gel sulfates, based on screening and pairing parameters, dictates the nature of the potential and ion distributions and drives the system to a steady state. The presence of the solvent as well as different ion valencies (in this paper, a divalent case) also affects the system, as alterations in ion pairing directly change the electrical potential that is created within the gel.

Future work can be performed to improve the transient model, as the errors discussed prior could potentially be eliminated with a more precise simulation. It is also noteworthy that, in both models of the transient system, the profiles measured have short wavelength "shakiness", as can be clearly seen in Figures 7 and 9, which depict the final concentrations of these systems. This is most likely due to a numerical instability in these calculations, and a higher-order model can reduce this noise and produce a more accurate representation of the Donnan potential and potential distributions.

Another concession is the simplicity of the steady state calculations. We assume that the glycocalyx is a linearly uniform layer of negative charge, which may not be an accurate representation of real life. Barbati and Kirby demonstrate that different profiles for the glycocalyx layer significantly affect the potential distribution, as depicted in Figure 12.[5] As a result, our models are admittedly simple models that will not apply if the glycocalyx is of a different configuration.

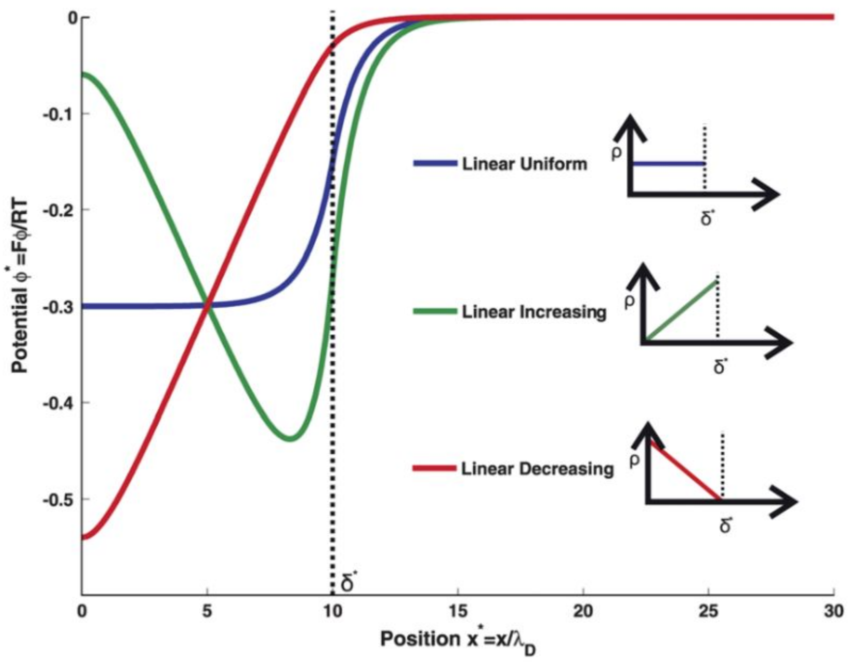

Figure 12. The electrical potential distribution of varying profiles of the glycocalyx layer.[5]

Finally, the simplification of the glycocalyx into a one-dimensional model ignores the complexity of ion pairing and variations within a three-dimensional volume of space, as aspects such as osmotic pressure may also play an important role in glycocalyx electrostatics. A three-dimensional model of the glycocalyx surrounded by a volume of salt using all-atom molecular dynamics may provide a more comprehensive model of the glycocalyx, beyond what our coarse-grained mean-field model can offer.

The models presented in this paper provide a foundation for future research on the glycocalyx, especially in regards to its electrostatic properties and ion-pairing behavior. We believe this theoretical and mathematical approach can serve as a basis for which these emergent biophysical properties can be studied and applied to biological research and biomedical engineering applications.

References

1. Sterling and Baker, Macromolecular Theory and Simulations, 27, 2, March 2018.

2. Linden SK, Sutton P, Karlsson NG, Korolik V, McGuckin MA. Mucins in the mucosal barrier to infection. Mucosal Immunol. 2008;1(3):183–97.

3. Electro-lyotropic equilibrium and the utility of ion-pair dissociation constants; James D. Sterling and Shenda M. Baker, Colloid and Interface Science Communications, 20, pp. 9-11, 2017

4. Iwahara, Esadze and Zandarashvili, Biomolecules, 2015, 5, 2435-2463

5. A. C. Barbati and B. J. Kirby , Soft Matter, 2012, 8 , 10598 -10613

Appendix A

Glycocalyx - Salt Interface Simulation

1. Steady State Equations

1.1 Salt Layer

The equation for the gel layer is a non-linear 2nd order ordinary differential equation.

$$\frac{d^2\phi_D(x)}{dx^2} = \frac{c_{i,o}}{c_{j,o}}\left(e^{\phi_D(x)} - e^{-\phi_D(x)}\right)$$

1.2 Gel Layer

The equation for the gel layer is a non-linear 2nd order ordinary differential equation.

$$\frac{d^2\phi_D(x)}{dx^2} = \frac{c_{i,o}}{c_{j,o}}\left(e^{\phi_D(x)} - e^{-\phi_D(x)}\right) + \left[1 + \frac{c_{i,o}}{K_{ij}}e^{-\phi_D(x)}\right]^{-1}$$

With boundary conditions of:

$$\phi_D(0) = \phi_s(0)$$

$$\frac{d^2\phi_D(x)}{dx^2}(x \to \infty) = 0$$

For the purpose of making the code readable, I am substituting the names of the variables such that this equation becomes:

$$y'' = S\left(e^y - e^{-y}\right) + \left[1 + P \cdot e^{-y}\right]^{-1}$$

And using the definition of the hyperbolic sine, this becomes further simplified to:

$$y'' = S \cdot 2\sinh y + \left[1 + P \cdot e^{-y}\right]^{-1}$$

And finally, to be able to use the scipy boundary value problem (bvp) solver, this equation must be re-written as a system of two 1st order ordinary differential equations.

$$z = y'$$

$$z' = y'' = S \cdot 2\sinh y + \left[1 + P \cdot e^{-y}\right]^{-1}$$

2. Steady State Simulation

2.1 'Jump' Conditions

Determine the value at $y(x \to \infty)$ using function solver

In [ ]:

```
# -------------------------
# import libraries
# -------------------------
import numpy as np
from scipy.integrate import solve_bvp
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
%matplotlib inline

# -------------------------
# define and determine constants
# -------------------------
# initialize constants selected to match the Matlab code
S = 0.1 # screening (c_i,o / c_j,o)
P = 0.1 # pairing   (c_i,o / K_ij)

# calculate y(x->inf) assuming y''(x->inf)=0
def gel(y):
    return S*(np.exp(y)-np.exp(-y)) + (1+P*np.exp(-y))**-1
y_inf = fsolve(gel,-2)[0]
print(y_inf)
```

2.2 Optimization of Intercept

Run code to find intercept such that the slopes of the salt layer and gel layer are matched.

In [ ]:

```python
# --------------------------
# Loop through potential intercepts
# and optimize for minimum difference in slope at interface
# --------------------------
min_d_slope = 100 # minimum difference in slope
min_intercept = 0 # intercept associated with min_d_slope
dx= 0.05
testrange = np.arange(-10, 10 + dx, dx)
for y0 in np.arange(0, y_inf, -0.001):
    # SALT LAYER
    # --------------------------
    def fun_s(x, y):
        z  = y[1]
        dz = S*2*np.sinh(y[0])
        return np.vstack((z, dz))

    def bc_s(ya, yb):
        return np.array([ya[0]-y0, yb[0]])

    xs = np.arange(0, 10+dx, dx)
    ys = -np.ones((2, xs.size))
    res = solve_bvp(fun_s, bc_s, xs, ys)
    ys = res.sol(xs)[0]
    slope_salt = (ys[1]-ys[0]) / (xs[1]-xs[0])

    # GEL LAYER
    # --------------------------
    def fun(x, y):
        z  = y[1]
        dz = S*2*np.sinh(y[0]) + (1+P*np.exp(-y[0]))**(-1)
        return np.vstack((z, dz))

    def bc(ya, yb):
        return np.array([ya[0]-y_inf, yb[0]-y0])

    x = np.arange(-10, 0+dx, dx)
    y = -np.ones((2, x.size))
    res = solve_bvp(fun, bc, x, y)
    y = res.sol(x)[0]
    slope_gel = (y[-1]-y[-2]) / (x[-1]-x[-2])

    # OPTIMIZATION
    # --------------------------
    d_slope = abs(slope_salt-slope_gel)
    if d_slope <= min_d_slope:
        min_d_slope = d_slope
        min_intercept = y0


print(min_d_slope, min_intercept)
```

2.3 Re-solve and Plot at Optimal Intercept

In [ ]:

```python
import numpy as np
from scipy.optimize import fsolve
from scipy.integrate import solve_bvp
import matplotlib.pyplot as plt
%matplotlib inline

dx = 0.05
y0 = min_intercept

# SALT LAYER
# --------------------------
def fun_s(x, y):
    z  = y[1]
    dz = S*2*np.sinh(y[0])
    return np.vstack((z, dz))

def bc_s(ya, yb):
    return np.array([ya[0]-y0, yb[0]])

xs = np.arange(0, 10+dx, dx)
ys = -np.ones((2, xs.size))
res = solve_bvp(fun_s, bc_s, xs, ys)
ys = res.sol(xs)[0]
# TANGENT LINE
# --------------------------
xi = np.linspace(-2, 2, num=2)
slope = (ys[1]-ys[0]) / (xs[1]-xs[0])
yi = slope*xi+y0


# GEL LAYER
# -------------------------
# Solve the differential equation
def fun(x, y):
    z  = y[1]
    dz = S*2*np.sinh(y[0]) + (1+P*np.exp(-y[0]))**(-1)
    return np.vstack((z, dz))

def bc(ya, yb):
    return np.array([ya[0]-y_inf, yb[0]-y0])

x = np.arange(-10, 0 + dx, dx)
y = -np.ones((2, x.size))
res = solve_bvp(fun, bc, x, y)
y = res.sol(x)[0]

# PLOT RESULTS
# --------------------------
plt.rcParams.update({'font.size': 17})
plt.figure(figsize=(16,9), dpi=500)
plt.plot(x, y, 'b', label = 'Gel Layer Potential')
plt.plot(xs, ys, 'r', label = 'Salt Layer Potential')
plt.plot(xi, yi, 'k:', label = 'Slope at the Interface')
plt.plot(0, y0, 'ko', ms=4) # marker at intercept
plt.plot([x[0], xs[-1]], [y0, y0], 'k:') # marker at intercept
plt.plot([x[0], xs[-1]], [y_inf, y_inf], 'k:')

plt.title('Glycocalyx Salt Interface Simulation: P='+ str(P) + ', S=' + str(S))
plt.xlabel('Dimensionless Distance from Interface')
plt.ylabel('Dimensionless Donnan Potential')
plt.legend()
plt.xlim(x[0], xs[-1])
plt.ylim(y[0]*1.1, 0)
plt.savefig('steady state.jpg')
```

3. Numerical Analysis

3.1 Measurement of Donnan Potential

In [ ]:

```python
Donnan = np.hstack((np.hstack((y[:-1], (y[-1] + ys[0])*0.5)), ys[1:]))

Donnan
```

3.2 Relevant Graphs of the Donnan Potential

In [ ]:

```python
DonnanPrime = np.gradient(Donnan, dx, edge_order = 2)
DonnanDoublePrime = np.gradient(DonnanPrime, dx, edge_order =2)

plt.rcParams.update({'font.size': 17})
plt.figure(figsize=(16,9), dpi=500)
plt.plot(testrange, DonnanDoublePrime, label=r" $\partial^2 y / dx^2$ ")
plt.plot(testrange, DonnanPrime, label = r" $\partial y / dx$ ")
plt.plot(testrange, Donnan, label = r" $y$ ")

plt.title('Glycocalyx Salt Interface Simulation: P='+ str(P) + ', S=' + str(S))
plt.legend()
plt.xlabel('Dimensionless Distance from Interface')
plt.ylabel('Dimensionless Numerical Value')
plt.xlim(testrange[0], testrange[-1])
plt.savefig('steady state derivs.jpg')
```

3.3 Ion Concentrations at Steady State

In [ ]:

```
Cs0 = 10
n = np.size(testrange)//2
m = np.size(testrange) - n - 1
Cc = Cs0*S*np.exp(-Donnan)
Ca = Cs0*S*np.exp(Donnan)
Cs = Cs0*(1+P*np.exp(-Donnan[:n+1]))**-1
Cs = np.hstack((Cs, np.zeros(m)))

plt.rcParams.update({'font.size': 17})
plt.figure(figsize=(16, 9), dpi=500)

plt.plot(testrange, Cs, color='C0', label="[s]-")
plt.plot(testrange, Cc, color='C1', label="[c]+")
plt.plot(testrange, Ca, color='C2', label="[a]-")
plt.xlabel('Dimensionless Distance from Interface')
plt.ylabel('Dimensionless Numerical Value')
plt.title("Final Concentrations of the Steady State")
plt.legend()
plt.xlim(testrange[0], testrange[-1])
plt.ylim(0, Cs0)
plt.savefig('steady state results concentrations.jpg')
```

In [ ]:

Variation of S and P in Steady State Model

In [ ]:
```
import numpy as np
from mpl_toolkits import mplot3d
from scipy.optimize import fsolve
from scipy.integrate import solve_bvp
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import MultipleLocator
%matplotlib inline

plt.rc('font',size=17)
plt.rc('axes',labelsize=17)

s = np.logspace(-2, 2, 20)
p = np.logspace(-2, 2, 20)
def phi(S, P):
    def gel(y):
        return S*(np.exp(y)-np.exp(-y)) + (1+P*np.exp(-y))**-1
    return fsolve(gel,-2)[0]

S, P = np.meshgrid(s, p)
PHI_ = np.array([phi(S_, P_) for S_, P_ in zip(np.ravel(S), np.ravel(P))])
PHI = PHI_.reshape(S.shape)

fig = plt.figure(figsize=(16, 9), dpi=500)
ax = fig.add_subplot(111, projection = '3d', title="Variation of S and P\n")
ax.plot_surface(np.log10(P), np.log10(S), PHI, cmap=cm.jet, alpha = 0.75)
ax.set_xlabel('\n\nlog10 of Pairing Constant, P')
ax.set_ylabel('\n\nlog10 of Screening Constant, S')
ax.set_zlabel('\n\nDonnan Potential, y')
for t in ax.xaxis.get_major_ticks(): t.label.set_fontsize(15)
for t in ax.yaxis.get_major_ticks(): t.label.set_fontsize(15)
for t in ax.zaxis.get_major_ticks(): t.label.set_fontsize(15)
ax.invert_xaxis()


ax.view_init(azim=-45)
ax.zaxis.set_major_locator(MultipleLocator(0.5))

#plt.xticks([0.1, 1])
#plt.yticks([0.1, 1])
contours = ax.contour(np.log10(P), np.log10(S), PHI, zdir='z', offset=np.min(PHI), cmap=cm.jet)
ax.clabel(contours, inline = True)

plt.show()
plt.savefig('S and P.jpg')

#\u03C6
```

In [ ]:

In [ ]:

Appendix B

Poisson-Nernst-Planck (PNP) Transient Model

Poisson equation describing the electric potential gradient is:

$$\nabla \cdot (-\epsilon\epsilon_0 \nabla\phi) = F\sum_{i=1}^{N} z_i c_i$$

Nernst-Planck equation (including eletrophoretic and diffusivity terms) :

$$\frac{\partial c_i}{\partial t} = \nabla \cdot \left(\frac{z_i}{|z_i|}u_i c_i \nabla\phi + D_i \nabla c_i\right) + \sum_{j=1}^{N}\left(-k_{aij}c_i c_j + k_{dij}c_{ij}\right)$$

We would like to create a 1-dimensional, transient model with three species, sulfate, a cation, and an anion. On the left hand side is a gel with the sulfate immobilized. On the right is a free solution with only the cations and anions.

Under our conditions, the Poisson equation becomes:

$$\begin{aligned} -\epsilon\epsilon_0 \frac{\partial^2\phi}{\partial x^2} &= F\left(-C_s + C_c - C_a\right) \\ \frac{\partial^2\phi}{\partial x^2} &= \frac{F}{-\epsilon\epsilon_0}(-C_s + C_c - C_a) \end{aligned}$$

Nondimensionalization brings:

$$\frac{\partial^2 y}{\partial x^2} = \frac{F}{-\epsilon\epsilon_0}\frac{(-C_s + C_c - C_a)}{c_{j,0}}$$

The Nernst-Planck equation on the gel side becomes:

$$\begin{aligned} \frac{\partial C_s}{\partial t} &= && -k_a C_s C_c + k_d(C_{j,0} - C_s) \\ \frac{\partial C_c}{\partial t} &= \frac{\partial}{\partial x}\left(+\mu_c C_c \frac{\partial\phi}{\partial x} + D_c \frac{\partial C_c}{\partial x}\right) && -k_a C_c C_s + k_d(C_{j,0} - C_s) \\ \frac{\partial C_a}{\partial t} &= \frac{\partial}{\partial x}\left(-\mu_a C_a \frac{\partial\phi}{\partial x} + D_a \frac{\partial C_a}{\partial x}\right) && \end{aligned}$$

Where $C_{j,0}$ is the total concentration of sulfates, bound and unbound. Therefore the difference, $(C_{j,0} - C_s)$, represents the number of bound cation-sulfate pairs.

We apply the spacial derivative to the electrophoretic and diffusion terms resulting in:

$$\begin{aligned} \frac{\partial C_s}{\partial t} &= && -k_a C_s C_c + k_d(C_{j,0} - C_s) \\ \frac{\partial C_c}{\partial t} &= +\mu_c C_c \frac{\partial^2\phi}{\partial x^2} + \mu_c \frac{\partial C_c}{\partial x}\frac{\partial\phi}{\partial x} + D_c \frac{\partial^2 C_c}{\partial x^2} && -k_a C_c C_s + k_d(C_{j,0} - C_s) \\ \frac{\partial C_a}{\partial t} &= -\mu_a C_a \frac{\partial^2\phi}{\partial x^2} - \mu_a \frac{\partial C_a}{\partial x}\frac{\partial\phi}{\partial x} + D_a \frac{\partial^2 C_a}{\partial x^2} && \end{aligned}$$

The Nernst-Planck equation on the salt solution side, given there are no sulfate bonding sites ($C_s = 0$), becomes:

$$\begin{aligned} \frac{\partial C_s}{\partial t} &= 0 \\ \frac{\partial C_c}{\partial t} &= +\mu_c C_c \frac{\partial^2\phi}{\partial x^2} + \mu_c \frac{\partial C_c}{\partial x}\frac{\partial\phi}{\partial x} + D_c \frac{\partial^2 C_c}{\partial x^2} \\ \frac{\partial C_a}{\partial t} &= -\mu_a C_a \frac{\partial^2\phi}{\partial x^2} - \mu_a \frac{\partial C_a}{\partial x}\frac{\partial\phi}{\partial x} + D_a \frac{\partial^2 C_a}{\partial x^2} \end{aligned}$$

The variables are:

$$\begin{aligned} \phi &\equiv \text{Electric potential} \\ \epsilon &\equiv \text{permittivity} \\ \epsilon_0 &\equiv \text{permittivity in vacuum} \\ F &\equiv \text{Faraday's constant} \\ C &\equiv \text{concentration} \\ D &\equiv \text{diffusivity} \\ \mu &\equiv \text{eletrophoretic mobility} \\ k &\equiv \text{rate constant} \\ c &\equiv \text{cation} \\ a &\equiv \text{anion} \\ s &\equiv \text{sulfate} \end{aligned}$$

1. Transient Simulation

1.1 Initial Conditions

```
In [ ]:  # Libraries
         #------------------------
         import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline


         # Space
         #------------------------
         n = 200 # number of mesh points on gel side (includes x[n] = 0, n+1 points total)
         m = 200 # number of mesh points on salt side (includes x[n+1] = dx, m points total)
         dx = 0.05
         x = np.hstack((np.arange(-n*dx, dx, dx), np.arange(dx, m*dx + dx, dx)))

         # Time
         #------------------------
         dt = 0.0005
         time = np.arange(0, 20, dt)

         print(dt/dx**2, "should be less than 0.5 if it is to be stable.")

         # Model Parameters
         #------------------------
         # constants
         F = 1
         e = 1
         e0 = 1
         ua = 1
         uc = 1
         Da = 1
         Dc = 1
         k1 = 1
         k2 = 10
         K = k2/k1
         P = 0.1
         S = 0.1
         Cc0 = P*K
         Cs0 = Cc0/S



         # initial condition (copied from steady-state)

         Ph0 = [-1.83983590e+00, -1.83983034e+00, -1.83982476e+00, -1.83981917e+00,
                -1.83981353e+00, -1.83980785e+00, -1.83980210e+00, -1.83979628e+00,
                -1.83979038e+00, -1.83978437e+00, -1.83977825e+00, -1.83977200e+00,
                -1.83976561e+00, -1.83975906e+00, -1.83975235e+00, -1.83974545e+00,
                -1.83973835e+00, -1.83973104e+00, -1.83972350e+00, -1.83971571e+00,
                -1.83970765e+00, -1.83969931e+00, -1.83969067e+00, -1.83968172e+00,
                -1.83967242e+00, -1.83966276e+00, -1.83965272e+00, -1.83964228e+00,
                -1.83963141e+00, -1.83962009e+00, -1.83960829e+00, -1.83959600e+00,
                -1.83958317e+00, -1.83956979e+00, -1.83955583e+00, -1.83954125e+00,
                -1.83952602e+00, -1.83951010e+00, -1.83949347e+00, -1.83947609e+00,
                -1.83945792e+00, -1.83943891e+00, -1.83941903e+00, -1.83939823e+00,
                -1.83937647e+00, -1.83935370e+00, -1.83932986e+00, -1.83930492e+00,
                -1.83927880e+00, -1.83925146e+00, -1.83922283e+00, -1.83919286e+00,
                -1.83916146e+00, -1.83912859e+00, -1.83909415e+00, -1.83905809e+00,
                -1.83902031e+00, -1.83898074e+00, -1.83893928e+00, -1.83889586e+00,
                -1.83885036e+00, -1.83880269e+00, -1.83875275e+00, -1.83870042e+00,
                -1.83864560e+00, -1.83858815e+00, -1.83852796e+00, -1.83846489e+00,
                -1.83839880e+00, -1.83832954e+00, -1.83825697e+00, -1.83818092e+00,
                -1.83810123e+00, -1.83801772e+00, -1.83793020e+00, -1.83783849e+00,
                -1.83774238e+00, -1.83764166e+00, -1.83753611e+00, -1.83742550e+00,
                -1.83730957e+00, -1.83718809e+00, -1.83706077e+00, -1.83692734e+00,
                -1.83678751e+00, -1.83664097e+00, -1.83648739e+00, -1.83632644e+00,
                -1.83615776e+00, -1.83598098e+00, -1.83579572e+00, -1.83560156e+00,
                -1.83539807e+00, -1.83518482e+00, -1.83496133e+00, -1.83472711e+00,
                -1.83448165e+00, -1.83422440e+00, -1.83395480e+00, -1.83367226e+00,
                -1.83337615e+00, -1.83306583e+00, -1.83274062e+00, -1.83239980e+00,
                -1.83204263e+00, -1.83166831e+00, -1.83127603e+00, -1.83086493e+00,
                -1.83043411e+00, -1.82998262e+00, -1.82950947e+00, -1.82901363e+00,
                -1.82849402e+00, -1.82794949e+00, -1.82737885e+00, -1.82678086e+00,
                -1.82615421e+00, -1.82549753e+00, -1.82480938e+00, -1.82408827e+00,
                -1.82333262e+00, -1.82254078e+00, -1.82171103e+00, -1.82084157e+00,
                -1.81993049e+00, -1.81897583e+00, -1.81797549e+00, -1.81692732e+00,
                -1.81582904e+00, -1.81467826e+00, -1.81347250e+00, -1.81220914e+00,
                -1.81088546e+00, -1.80949859e+00, -1.80804554e+00, -1.80652318e+00,
                -1.80492823e+00, -1.80325726e+00, -1.80150668e+00, -1.79967274e+00,
                -1.79775150e+00, -1.79573887e+00, -1.79363053e+00, -1.79142200e+00,
                -1.78910856e+00, -1.78668531e+00, -1.78414710e+00, -1.78148856e+00,
                -1.77870406e+00, -1.77578773e+00, -1.77273344e+00, -1.76953477e+00,
                -1.76618502e+00, -1.76267718e+00, -1.75900396e+00, -1.75515770e+00,
                -1.75113043e+00, -1.74691384e+00, -1.74249923e+00, -1.73787754e+00,
                -1.73303931e+00, -1.72797466e+00, -1.72267332e+00, -1.71712454e+00,
                -1.71131714e+00, -1.70523947e+00, -1.69887938e+00, -1.69222422e+00,
                -1.68526081e+00, -1.67797545e+00, -1.67035387e+00, -1.66238121e+00,
                -1.65404205e+00, -1.64532033e+00, -1.63619937e+00, -1.62666185e+00,
                -1.61668977e+00, -1.60626445e+00, -1.59536653e+00, -1.58397591e+00,
                -1.57207176e+00, -1.55963250e+00, -1.54663579e+00, -1.53305850e+00,
                -1.51887670e+00, -1.50406568e+00, -1.48859987e+00, -1.47245290e+00,
                -1.45559754e+00, -1.43800571e+00, -1.41964849e+00, -1.40049610e+00,
                -1.38051787e+00, -1.35968228e+00, -1.33795697e+00, -1.31530868e+00,
                -1.29170332e+00, -1.26710596e+00, -1.24148082e+00, -1.21479130e+00,
                -1.18700000e+00, -1.15923370e+00, -1.13218594e+00, -1.10583330e+00,
                -1.08015345e+00, -1.05512507e+00, -1.03072781e+00, -1.00694222e+00,
                -9.83749656e-01, -9.61132297e-01, -9.39073044e-01, -9.17555501e-01,
                -8.96563925e-01, -8.76083197e-01, -8.56098779e-01, -8.36596689e-01,
                -8.17563467e-01, -7.98986149e-01, -7.80852241e-01, -7.63149694e-01,
                -7.45866883e-01, -7.28992587e-01, -7.12515966e-01, -6.96426545e-01,
                -6.80714199e-01, -6.65369131e-01, -6.50381863e-01, -6.35743219e-01,
                -6.21444312e-01, -6.07476531e-01, -5.93831529e-01, -5.80501216e-01,
                -5.67477742e-01, -5.54753492e-01, -5.42321074e-01, -5.30173311e-01,
                -5.18303234e-01, -5.06704073e-01, -4.95369248e-01, -4.84292365e-01,
                -4.73467205e-01, -4.62887723e-01, -4.52548037e-01, -4.42442426e-01,
                -4.32565321e-01, -4.22911303e-01, -4.13475095e-01, -4.04251559e-01,
                -3.95235693e-01, -3.86422622e-01, -3.77807597e-01, -3.69385993e-01,
                -3.61153300e-01, -3.53105124e-01, -3.45237179e-01, -3.37545290e-01,
                -3.30025383e-01, -3.22673486e-01, -3.15485725e-01, -3.08458320e-01,
                -3.01587585e-01, -2.94869924e-01, -2.88301825e-01, -2.81879863e-01,
                -2.75600696e-01, -2.69461060e-01, -2.63457771e-01, -2.57587718e-01,
                -2.51847866e-01, -2.46235252e-01, -2.40746980e-01, -2.35380224e-01,
                -2.30132224e-01, -2.25000284e-01, -2.19981770e-01, -2.15074111e-01,
                -2.10274794e-01, -2.05581364e-01, -2.00991424e-01, -1.96502629e-01,
                -1.92112692e-01, -1.87819374e-01, -1.83620491e-01, -1.79513905e-01,
       -1.75497529e-01, -1.71569323e-01, -1.67727293e-01, -1.63969490e-01,
       -1.60294009e-01, -1.56698986e-01, -1.53182603e-01, -1.49743078e-01,
       -1.46378673e-01, -1.43087687e-01, -1.39868456e-01, -1.36719354e-01,
       -1.33638792e-01, -1.30625215e-01, -1.27677102e-01, -1.24792968e-01,
       -1.21971358e-01, -1.19210851e-01, -1.16510056e-01, -1.13867613e-01,
       -1.11282193e-01, -1.08752493e-01, -1.06277242e-01, -1.03855195e-01,
       -1.01485134e-01, -9.91658664e-02, -9.68962283e-02, -9.46750787e-02,
       -9.25013019e-02, -9.03738064e-02, -8.82915238e-02, -8.62534089e-02,
       -8.42584387e-02, -8.23056122e-02, -8.03939496e-02, -7.85224919e-02,
       -7.66903005e-02, -7.48964567e-02, -7.31400610e-02, -7.14202328e-02,
       -6.97361102e-02, -6.80868491e-02, -6.64716231e-02, -6.48896228e-02,
       -6.33400557e-02, -6.18221455e-02, -6.03351321e-02, -5.88782707e-02,
       -5.74508317e-02, -5.60521006e-02, -5.46813769e-02, -5.33379745e-02,
       -5.20212210e-02, -5.07304573e-02, -4.94650375e-02, -4.82243283e-02,
       -4.70077089e-02, -4.58145707e-02, -4.46443166e-02, -4.34963614e-02,
       -4.23701308e-02, -4.12650614e-02, -4.01806007e-02, -3.91162063e-02,
       -3.80713459e-02, -3.70454972e-02, -3.60381471e-02, -3.50487922e-02,
       -3.40769377e-02, -3.31220979e-02, -3.21837956e-02, -3.12615617e-02,
       -3.03549353e-02, -2.94634634e-02, -2.85867005e-02, -2.77242084e-02,
       -2.68755563e-02, -2.60403201e-02, -2.52180825e-02, -2.44084328e-02,
       -2.36109665e-02, -2.28252854e-02, -2.20509968e-02, -2.12877142e-02,
       -2.05350564e-02, -1.97926474e-02, -1.90601165e-02, -1.83370979e-02,
       -1.76232307e-02, -1.69181583e-02, -1.62215288e-02, -1.55329943e-02,
       -1.48522112e-02, -1.41788395e-02, -1.35125432e-02, -1.28529896e-02,
       -1.21998495e-02, -1.15527969e-02, -1.09115090e-02, -1.02756656e-02,
       -9.64494934e-03, -9.01904554e-03, -8.39764183e-03, -7.78042811e-03,
       -7.16709639e-03, -6.55734062e-03, -5.95085653e-03, -5.34734152e-03,
       -4.74649444e-03, -4.14801551e-03, -3.55160613e-03, -2.95696870e-03,
       -2.36380657e-03, -1.77182378e-03, -1.18072498e-03, -5.90215254e-04,
        8.88325803e-20]
Ph0 = np.array(Ph0)

Cc = Cs0*S*np.exp(-Ph0)
Ca = Cs0*S*np.exp(Ph0)
#Disturbed Transient Conditions
#Cc = np.flip(Cc)
#Ca = np.flip(Ca)
Cs = Cs0*(1+P*np.exp(-Ph0[:n+1]))**-1
Cs = np.hstack((Cs, np.zeros(m)))
totalCs = np.hstack((np.ones(n+1)*Cs0, np.ones(m)*0))
```

1.2 Evolution Simulation

In [ ]:

```
#------------------------
# SIMULATION STARTS HERE
#------------------------
dCsdt = np.empty(n+m+1)
dCcdt = np.empty(n+m+1)
dCadt = np.empty(n+m+1)

plt.rcParams.update({'font.size': 17})
plt.figure(figsize=(16, 9), dpi = 500)
plt.xlabel('Dimensionless Distance from Interface')
plt.ylabel('Dimensionless Donnan Potential')
plt.title('Donnan Potential Distribution of Glycocalyx Salt Interface Simulation: P='+ str(P) + ', S=' + str(S))

for j in range(len(time)):

    if (j*dt)%1 == 0:
        print(j/2000)
    # calculate the electric field and gradient
    # -----------------------------------------
    d2Phdx2 = F/(-e*e0)*(-Cs+Cc-Ca)/Cs0
    dPhdx = np.cumsum(d2Phdx2)*dx

    # calculate the gradients of each concentration
    # ------------------------------------------
    dCcdx = np.gradient(Cc, dx, edge_order=2)
    dCcdx[0] = 0
    dCcdx[-1] = 0

    d2Ccdx2 = np.gradient(dCcdx, dx, edge_order=2)
    d2Ccdx2[0] = 0
    d2Ccdx2[-1] = 0

    dCadx = np.gradient(Ca, dx, edge_order=2)
    dCadx[0] = 0
    dCadx[-1] = 0

    d2Cadx2 = np.gradient(dCadx, dx, edge_order=2)
    d2Cadx2[0] = 0
    d2Cadx2[-1] = 0

    dCsdt = -k1*Cs*Cc+k2*(totalCs-Cs)
    dCcdt = +uc*Cc*d2Phdx2+uc*dCcdx*dPhdx+Dc*d2Ccdx2+dCsdt
    dCadt = -ua*Ca*d2Phdx2-ua*dCadx*dPhdx+Da*d2Cadx2


    # calculate further derivatives
    # -----------------------------------------
    d2Ccdt2 = np.gradient(dCcdt, dt, edge_order=2)
    d2Ccdt2[-1] = 0

    d2Cadt2 = np.gradient(dCadt, dt, edge_order=2)
    d2Cadt2[-1] = 0


    # Update values to next time step
    # -----------------------------------------
    Cs = Cs + dCsdt*dt
    Cc = Cc + dCcdt*dt + 0.5*d2Ccdt2*dt**2
    Ca = Ca + dCadt*dt + 0.5*d2Cadt2*dt**2

    for i in range(len(x)):
        Cs[i] = max(0, Cs[i])
        Cc[i] = max(0, Cc[i])
        Ca[i] = max(0, Ca[i])


    if j%500 == 0:
        b = (1-j/len(time)) *0.8
        g =0
        r = j/len(time)
        c = (r, g, b)
        dPhdx[:n+1] -= np.min(dPhdx[:n+1])
        dPhdx[n+1:] -= np.min(dPhdx[n+1:])
        Ph = np.cumsum(dPhdx)*dx
        Ph = Ph-Ph[-1]
        print(Ph[0])
        plt.plot(x, Ph, color=c, label = r" $Donnan Potential Distribution$ ")
plt.savefig('transient results wave.jpg')
```

2. Final System

2.1 Final Concentrations

In [ ]:

```
# Plot Results
#------------------------
# Concentrations
plt.rcParams.update({'font.size': 17})
plt.figure(figsize=(16, 9), dpi=500)

plt.plot(x, Cs, color='C0', label="[s]-")
plt.plot([x[0], 0], [Cs0, Cs0], 'k:', label="[s]_all")
plt.plot(x, Cc, color='C1', label="[c]+")
plt.plot(x, Ca, color='C2', label="[a]-")
plt.xlabel('Dimensionless Distance from Interface')
plt.ylabel('Dimensionless Numerical Value')
plt.title("Final Concentrations of the Initialized Steady State")
plt.legend()
plt.savefig('transient results concentrations.jpg')
```

2.2 Final Potential

In [ ]:
```
# Compute the Potential
dPhdx[:n+1] -= np.min(dPhdx[:n+1])
dPhdx[n+1:] -= np.min(dPhdx[n+1:])
Ph = np.cumsum(dPhdx)*dx
Ph = Ph-Ph[-1]

xi = np.linspace(-2,2, num=2)
slope = (Ph[n+1]-Ph[n])/dx
yi = slope*xi+Ph[n]

# Plot the Potential with Annotations
plt.rcParams.update({'font.size': 17})
plt.figure(figsize=(16, 9), dpi=500)

plt.plot(x[:n+1], Ph[:n+1], 'b', label='Gel Layer Potential')
plt.plot(x[n:], Ph[n:], 'r', label='Salt Layer Potential')
plt.plot(xi, yi, 'k:', label = 'Slope at the Interface')
plt.plot(0, Ph[n], 'ko', ms=4)
plt.plot([x[0], x[-1]], [Ph[n], Ph[n]], 'k:')
plt.plot([x[0], x[-1]], [np.min(Ph), np.min(Ph)], 'k:')
plt.xlabel('Dimensionless Distance from Interface')
plt.ylabel('Dimensionless Numerical Value')
plt.title("Final Donnan Potential Distribution")
plt.legend()
plt.xlim(x[0], x[-1])
plt.ylim(-2, 0.01)

plt.savefig('transient results derivs.jpg')
```

2.3 Relevant Graphs of Donnan Potential

In [ ]:
```
# Plot Potential, First Derivative, and Second Derivative
plt.rcParams.update({'font.size': 17})
plt.figure(figsize=(16, 9), dpi=500)
plt.title('Donnan Potential Distribution of Glycocalyx Salt Interface Simulation: P='+ str(P) + ', S=' + str(S))
plt.plot(x, d2Phdx2, label=r" $\partial^2 y / dx^2$ ")
plt.plot(x, dPhdx, label = r" $\partial y / dx$ ")
plt.plot(x, Ph, label = r" $y$ ")
plt.xlabel('Dimensionless Distance from Interface')
plt.ylabel('Dimensionless Numerical Value')
plt.legend()
plt.xlim(x[0], x[-1])

plt.savefig('transient results.jpg')
```

In [ ]:

In [ ]:

Appendix C

Variation of Constants in Solvent-Separated Model

Variation of P and Q at Constant S

In [ ]:

```
import numpy as np
from mpl_toolkits import mplot3d
from scipy.optimize import fsolve
from scipy.integrate import solve_bvp
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import MultipleLocator
%matplotlib inline

plt.rc('font',size=17)
plt.rc('axes',labelsize=17)

S = 0.1 # constant
p = np.logspace(-2, 2, 20)
q = np.logspace(-2, 2, 20)
def phi(S, P, Q):
    def gel(y):
        return S*(np.exp(y)-np.exp(-y)) + (1+P*np.exp(-y)*(1+Q))**-1
    return fsolve(gel,-2)[0]

P, Q = np.meshgrid(p, q)
PHI_ = np.array([phi(S, P_, Q_) for P_, Q_ in zip(np.ravel(P), np.ravel(Q))])
PHI = PHI_.reshape(P.shape)

fig = plt.figure(figsize=(16, 9), dpi=500)
ax = fig.add_subplot(111, projection = '3d', title="Variation of P and Q at Constant S = 0.1\n")
ax.plot_surface(np.log10(Q), np.log10(P), PHI, cmap=cm.jet, alpha = 0.75)
ax.set_xlabel('\n\nlog10 of CIP Constant, Q')
ax.set_ylabel('\n\nlog10 of SSIP Constant, P')
ax.set_zlabel('\n\nDonnan Potential, y')
for t in ax.xaxis.get_major_ticks(): t.label.set_fontsize(15)
for t in ax.yaxis.get_major_ticks(): t.label.set_fontsize(15)
for t in ax.zaxis.get_major_ticks(): t.label.set_fontsize(15)
ax.invert_xaxis()



ax.view_init(azim=-45)
ax.zaxis.set_major_locator(MultipleLocator(0.5))

#plt.xticks([0.1, 1])
#plt.yticks([0.1, 1])
ax.contour(np.log10(Q), np.log10(P), PHI, zdir='z', offset=np.min(PHI), cmap=cm.jet)


plt.show()
plt.savefig('P and Q.jpg')
```

In [ ]:

Appendix D

Variation of Divalent Constants

```
In [ ]:  import numpy as np
         from mpl_toolkits import mplot3d
         from scipy.optimize import fsolve
         from scipy.integrate import solve_bvp
         import matplotlib.pyplot as plt
         from matplotlib import cm
         from matplotlib.ticker import MultipleLocator
         %matplotlib inline

         plt.rc('font',size=17)
         plt.rc('axes',labelsize=17)
```

1. Constant S

```
In [ ]:  # cations are diavalent
         # variables involved
         S = 0.1 # constant = ca/cg
         c_g = 0.1 # concentration of anions in gel (C_jo)
         p1 = np.logspace(-2, 2, 20)
         p2 = np.logspace(-2, 2, 20)

         def phi(S, P1, P2):
             c_a0 = S*c_g # concentration of anion in salt
             c_c0 = c_a0/2 # concentration of cation in salt
             def alpha(S, P1, P2, y):
                 return 2*c_c0*np.exp(-y)*P1*P2
             def beta(S, P1, P2, y):
                 return 1 + c_c0*np.exp(-y)*P1
             def free(y):
                 return c_a0*(-np.exp(y)+np.exp(-2*y))
             def gel(y):
                 a = alpha(S, P1, P2, y)
                 b = beta(S, P1, P2, y)
                 return (-free(y) + (-b + (b**2 + 4*c_g*a)**(0.5))/(2*a))/c_g
             return fsolve(gel,-1)[0]

         P1, P2 = np.meshgrid(p1, p2)
         PHI_ = np.array([phi(S, P1_, P2_) for P1_, P2_ in zip(np.ravel(P1), np.ravel(P2))])
         PHI = PHI_.reshape(P1.shape)

         fig = plt.figure(figsize=(16, 9), dpi=500)
         ax = fig.add_subplot(111, projection = '3d', title="Variation of P1 and P2 at Constant S = 0.1\n")
         ax.plot_surface(np.log10(P1), np.log10(P2), PHI, cmap=cm.jet, alpha = 0.75)
         ax.set_xlabel('\n\nlog10 of P1')
         ax.set_ylabel('\n\nlog10 of P2')
         ax.set_zlabel('\n\nDonnan Potential, y')
         for t in ax.xaxis.get_major_ticks(): t.label.set_fontsize(15)
         for t in ax.yaxis.get_major_ticks(): t.label.set_fontsize(15)
         for t in ax.zaxis.get_major_ticks(): t.label.set_fontsize(15)
         ax.invert_xaxis()


         ax.view_init(azim=-45)
         ax.zaxis.set_major_locator(MultipleLocator(0.25))

         contours = ax.contour(np.log10(P1), np.log10(P2), PHI, zdir='z', offset=np.min(PHI), cmap=cm.jet)
         ax.clabel(contours, inline = True)

         plt.show()
         plt.savefig('constant s.jpg')

         #\u03C6
```

2. Constant P1

```
In [ ]:  # variables involved
         s = np.logspace(-1, 3, 20) # constant = ca/cg
         c_g = 0.1 # concentration of anions in gel (C_jo)
         P1 = 0.1
         p2 = np.logspace(-1, 3, 20)

         S, P2 = np.meshgrid(s, p2)
         PHI_ = np.array([phi(S_, P1, P2_) for S_, P2_ in zip(np.ravel(S), np.ravel(P2))])
         PHI = PHI_.reshape(P2.shape)

         fig = plt.figure(figsize=(16, 9))
         ax = fig.add_subplot(111, projection = '3d', title="Variation of S and P2 at Constant P1 = 0.1\n")
         ax.plot_surface(np.log10(S), np.log10(P2), PHI, cmap=cm.jet, alpha = 0.75)
         ax.set_xlabel('\n\nlog10 of S')
         ax.set_ylabel('\n\nlog10 of P2')
         ax.set_zlabel('\n\nDonnan Potential, y')
         for t in ax.xaxis.get_major_ticks(): t.label.set_fontsize(15)
         for t in ax.yaxis.get_major_ticks(): t.label.set_fontsize(15)
         for t in ax.zaxis.get_major_ticks(): t.label.set_fontsize(15)
         ax.invert_xaxis()


         ax.view_init(azim=-45)
         ax.zaxis.set_major_locator(MultipleLocator(0.25))

         contours = ax.contour(np.log10(S), np.log10(P2), PHI, zdir='z', offset=np.min(PHI), cmap=cm.jet)
         ax.clabel(contours, inline = True)

         plt.show()
         plt.savefig('constant p1.jpg')
```

3. Constant P2

In [ ]:

```python
# variables involved
s = np.logspace(-1, 3, 20) # constant = ca/cg
c_g = 0.1 # concentration of anions in gel (C_jo)
p1 = np.logspace(-1, 3, 20)
P2 = 0.1

S, P1 = np.meshgrid(s, p1)
PHI_ = np.array([phi(S_, P1_, P2) for S_, P1_ in zip(np.ravel(S), np.ravel(P1))])
PHI = PHI_.reshape(P1.shape)

fig = plt.figure(figsize=(16, 9))
ax = fig.add_subplot(111, projection = '3d', title="Variation of S and P1 at Constant P2 = 0.1\n")
ax.plot_surface(np.log10(S), np.log10(P1), PHI, cmap=cm.jet, alpha = 0.75)
ax.set_xlabel('\n\nlog10 of S')
ax.set_ylabel('\n\nlog10 of P1')
ax.set_zlabel('\n\nDonnan Potential, y')
for t in ax.xaxis.get_major_ticks(): t.label.set_fontsize(15)
for t in ax.yaxis.get_major_ticks(): t.label.set_fontsize(15)
for t in ax.zaxis.get_major_ticks(): t.label.set_fontsize(15)
ax.invert_xaxis()


ax.view_init(azim=-45)
ax.zaxis.set_major_locator(MultipleLocator(0.25))

contours = ax.contour(np.log10(S), np.log10(P1), PHI, zdir='z', offset=np.min(PHI), cmap=cm.jet)
ax.clabel(contours, inline = True)

plt.show()
plt.savefig('constant p2.jpg')
```