RESEARCH ARTICLE

Algorithm for the Time-Propagation of the Radial Diffusion Equation Based on a Gaussian Quadrature

Dirk Gillespie*

Department of Molecular Biophysics and Physiology, Rush University Medical Center, 1750 West Harrison Street, Suite 1289, Chicago, Illinois, United States of America

* dirk_gillespie@rush.edu

OPEN ACCESS

Citation: Gillespie D (2015) Algorithm for the Time-Propagation of the Radial Diffusion Equation Based on a Gaussian Quadrature. PLoS ONE 10(7): e0132273. doi:10.1371/journal.pone.0132273

Editor: Francesco Pappalardo, University of Catania, ITALY

Received: March 25, 2015

Accepted: June 11, 2015

Published: July 24, 2015

Copyright: © 2015 Dirk Gillespie. This is an open access article distributed under the terms of the [Creative Commons Attribution License](https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Data Availability Statement: All relevant data are within the paper and its Supporting Information files.

Funding: This work was done during the term of Award 14GRNT20380691 from the American Heart Association (www.heart.org). The funder had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The author has declared that no competing interests exist.

Abstract

The numerical integration of the time-dependent spherically-symmetric radial diffusion equation from a point source is considered. The flux through the source can vary in time, possibly stochastically based on the concentration produced by the source itself. Fick's one-dimensional diffusion equation is integrated over a time interval by considering a source term and a propagation term. The source term adds new particles during the time interval, while the propagation term diffuses the concentration profile of the previous time step. The integral in the propagation term is evaluated numerically using a combination of a new diffusion-specific Gaussian quadrature and interpolation on a diffusion-specific grid. This attempts to balance accuracy with the least number of points for both integration and interpolation. The theory can also be extended to include a simple reaction-diffusion equation in the limit of high buffer concentrations. The method is unconditionally stable. In fact, not only does it converge for any time step Δt , the method offers one advantage over other methods because Δt can be arbitrarily large; it is solely defined by the timescale on which the flux source turns on and off.

Introduction

Diffusion is one of the most classic problems in physics, with the governing laws of Fick dating back more than 150 years [1, 2] and a number of books on the subject [3, 4, 5]. From both the mathematical and numerical points of view, the diffusion equation—especially the spherically-symmetric version considered here—is very well studied; its mathematical properties are known and a number of numerical techniques for solving it exist (e.g., explicit-time methods like forward-time/central-space (FTCS), implicit-time methods like backward-time/central-space (BTCS) and Crank-Nicolson, and more sophisticated ones [6]). While diffusion has been studied experimentally and theoretically for a very long time, it is still of central importance in many areas today. For example, it plays vital roles in biology [7] and new technologies like nanofluidics [8].

For these applications, the efficient numerical solution of the diffusion equation is fundamental to understanding the physics underlying these new applications. One important

example is calcium-induced calcium release (CICR) in cardiac muscle, where an array of Ca^{2+} -selective ion channels open and close with a probability that depends on the local Ca^{2+} concentration, which comes from the diffusion of Ca^{2+} through neighboring channels [9]. In this sense, the channel's open/closed state history affects its future state in a feedback mechanism. For such systems, the source flux is changing stochastically in time, depending on the on/off states of all the other flux sources. Moreover, since each channel is ~ 40 nm across, the calculation of a 10×10 array of these channels must be accurate over long distances (at least 560 nm) to understand the complex interactions.

Traditional numerical algorithms like BTCS and Crank-Nicolson divide the space coordinate into small slices to calculate numerical derivatives and produce a matrix equation that must be solved at each time step. Similarly, the conditionally-stable explicit-time methods like FTCS require a matrix/vector multiplication. Because the matrix is sparse, the number of flops for this solve is proportional to the number of grid points. These approaches are very valuable and easy to implement, but when the number of grid points reaches well into the thousands for large systems other approaches might be more computationally efficient.

Here, one alternative numerical technique to compute the concentration profile for radial Fickian diffusion from a point source is described. Specifically, the spherically-symmetric equation considered here is

$$\frac{\partial c(r, t)}{\partial t} = \frac{D}{r^2} \frac{\partial}{\partial r} \left(r^2 \frac{\partial c}{\partial r}(r, t) \right) + s(r, t) \quad (1)$$

with the initial condition

$$c(r, 0) = c_0, \quad (2)$$

which we take to be a constant. Here, $c(r, t)$ is the concentration (number per unit volume) of the diffusing particles and $s(r, t)$ is the flux per volume (number per unit volume per time) injected into the system. Here, a point source is considered. D is the diffusion coefficient and the variables r and t are the distance from the origin and time, respectively.

The purpose of this paper is describe a method to solve Eq (1) numerically for $c(r, t)$. Instead of discretizing the differential equation itself, the propagation algorithm described here is based on an integral version of the solution to Eq (1). This approach splits the solution into two components, the diffusion from the source during the time interval (whose exact solution is known) and the propagation of the particles released during previous time intervals. Here, we show that the propagation integral can be very efficiently calculated using a combination of (1) a new Gaussian quadrature that is specifically formulated for the diffusion equation and (2) interpolation with grid points whose locations are chosen based on the exact solution to the diffusion equation. This, then, minimizes both the number of points needed to evaluate the integral and the number of points needed to interpolate the results between grid points.

The propagation method offers advantages over more classical techniques. First, since no numerical time derivatives are used, the time step Δt does not have to be small. In the propagation method, Δt is determined by the source's timescale. For example, if the source flux varies smoothly, then Δt is chosen to be small to approximate that function. However, if the source turns on and off stochastically, then Δt should be the timescale that the source operates on; if, for example, it changes at most once a second, then Δt is 1 second. Second, since no numerical spatial derivatives are used with a constant Δr , very large numbers of grid points (and therefore very large tridiagonal matrices) are avoided. In the propagation method, a nonuniform grid is created that places points optimally for the diffusion problem and between those points interpolation is shown to be accurate.

Preliminary calculations show that a 10-point Gaussian quadrature and ~ 100 interpolation grid points can give very accurate results, even over long simulations using only a matrix/vector multiplication with a sparse matrix containing < 1400 nonzero entries. These calculations also show that it is possible to evaluate the error of the propagation technique. Moreover, the technique is fast enough to find an optimal set of simulation parameters that maximize accuracy and minimize calculation time. This error and parameter determination is part of the overhead of the simulation so that simulations of the system of interest can be done knowing that the error in the concentration is always within acceptable limits and that the run time is the shortest possible.

Theory

Propagation integral formulation

For the source flux in Eq (1) we consider a point source. When the point source has a constant flux j that does not vary in time so that

$$s(r, t) = j\delta(r)/4\pi r^2 \quad (3)$$

where $\delta(r)$ is the Dirac delta function, a simple closed form formula exists in terms of the complementary error function [3]:

$$c(r, t) = \frac{j}{4\pi D r} \operatorname{erfc}\left(\frac{r}{2\sqrt{Dt}}\right) \quad (4)$$

where

$$\operatorname{erfc}(x) = 1 - \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt. \quad (5)$$

However, when the source is not constant over time, Eq (1) must be solved numerically.

One way to attack this problem is via classic finite difference or finite elements methods. Methods like FTCS, BTCS, and Crank-Nicolson produce matrix equations to be solved for the concentration at previously-defined spatial grid points. The choice of grid points is crucial since minimizing the number of points is critical for fast calculation, but accuracy may suffer if the number is too small. The goal of this paper is to provide an alternative solution method that is not based on finite differences or finite elements. This approach propagates the concentration profile in time with a sparse-matrix/vector multiplication, as compared to solving a sparse-matrix equation. Since the grid points in this paper are based on a Gaussian quadrature and an interpolation scheme that are both specifically developed for this diffusion problem, the number of grid points and their locations are optimized in the sense of producing a high-order solution with a minimum number of points.

The approach we use here involves decomposing the concentration into two parts, a source part that adds new particles from the point source during the discretized time interval $T_n = (t_n, t_{n+1})$ and a propagation part that diffuses the concentration profile of the last time step through the time interval T_n [3]. That is,

$$c(r, t_{n+1}) = c_{\text{prop}}(r, t_{n+1}) + c_{\text{source}}(r, t_{n+1}) \quad (6)$$

where

$$c_{\text{prop}}(r, t_{n+1}) = \int_{\mathbb{R}^3} (4\pi D(t_{n+1} - t_n))^{-3/2} \exp\left(-\frac{|\mathbf{r} - \mathbf{r}'|^2}{4D(t_{n+1} - t_n)}\right) c(r', t_n) d\mathbf{r}' \quad (7)$$

and

$$c_{\text{source}}(r, t_{n+1}) = \chi_n j_n \int_{t_n}^{t_{n+1}} \int_{\mathbb{R}^3} (4\pi D (t_{n+1} - t'))^{-3/2} \exp\left(-\frac{|\mathbf{r} - \mathbf{r}'|^2}{4D(t_{n+1} - t')}\right) s(r', t') d\mathbf{r}' dt' \quad (8)$$

$$= \frac{\chi_n j_n}{4\pi D r} \operatorname{erfc}\left(\frac{r}{2\sqrt{D\Delta t}}\right) \quad (9)$$

where χ_n is 0 or 1 depending if the source is off or on during the time interval T_n , respectively, j_n is the flux during T_n , and $\Delta t = t_{n+1} - t_n$ is assumed to be the same for all n . These classic formulas may be found, for example, in Section 8.4 of the book by Barton [3]. (Note that for a half-space problem like flux through an ion channel on one side of a membrane, the denominator includes 2π not 4π because the surface area is half that of the whole sphere.)

Here, the source flux takes the form

$$s(r, t) = \chi(t) j(t) \delta(r) / 4\pi r^2 \quad (10)$$

and is assumed to be constant during the discretized time intervals:

$$j(t) = j_n \quad \forall t \in [t_n, t_{n+1}). \quad (11)$$

Similarly, χ_n , the on or off state of the source, is the same during the entire time interval $[t_n, t_{n+1})$. Note that here we do not assume that the χ_n are known beforehand; for that, an analytic solution of the diffusion equation exists (see Eq (54) later). Here, the on/off state of the source can vary due to random inputs that depend on the concentration profile produced by the source. For example, in CICR, the concentration profile produced by one channel affects the open state of its neighbors, which in turn produce Ca^{2+} profiles that affect the original channel.

It is also important to note that while we assume that j is constant over a time interval, that does not mean we are only considering fluxes that are either on or off or that j must be the same during all time intervals. Specifically, if the source flux is a smoothly varying function, the time intervals T_n should be chosen small enough so that $j(t)$ is well-approximated by the piecewise-constant function $j(t) = j_n \equiv j(t_n)$ if $t \in T_n$. In the notation used here, j_n is the flux during the n -th time interval and χ_n is a random variable that takes values 0 or 1. The χ function is convenient for the case of a randomly on or off source. For a smoothly varying source on the other hand, $\chi_n = 1$ for all n and the j_n are different for different n .

Since the source term is the concentration profile due to the *time-independent* source flux during the time interval T_n , Eq (9) is just Eq (4) applied to this time interval. In Eqs (7) and (8), the vectors $\mathbf{r}$ and $\mathbf{r}'$ (with lengths r and r' , respectively) are three-dimensional locations near the point source. The propagation integral can be simplified by using the relations

$$\begin{aligned} & \int_{\mathbb{R}^3} \exp\left(-\frac{|\mathbf{r} - \mathbf{r}'|^2}{4D\Delta t}\right) c(r', t_n) d\mathbf{r}' \\ &= 2\pi \int_0^\infty \int_0^\pi (r')^2 \sin(\theta') \exp\left(-\frac{r^2 + (r')^2 - 2rr' \cos(\theta')}{4D\Delta t}\right) c(r', t_n) d\theta' dr' \end{aligned} \quad (12)$$

and

$$\int_0^\pi \sin(\theta') \exp\left(\frac{rr' \cos(\theta')}{2D\Delta t}\right) d\theta' = \frac{4D\Delta t}{rr'} \sinh\left(\frac{rr'}{2D\Delta t}\right) \quad (13)$$

to give

$$c_{\text{prop}}(r, t_{n+1}) = \frac{1}{\sqrt{\pi D \Delta t}} \frac{1}{r} \exp\left(-\frac{r^2}{4D\Delta t}\right) \int_0^\infty c(r', t_n) \cdot r' \exp\left(-\frac{(r')^2}{4D\Delta t}\right) \sinh\left(\frac{rr'}{2D\Delta t}\right) dr'. \quad (14)$$

The accurate integration of this integral is computationally difficult because it requires knowing the function $c(r', t_n)$ for all r' from 0 to ∞ . This is generally difficult because one needs at least some prior knowledge about a reasonable finite upper limit and about acceptable grid spacings for the r' . One purpose of this paper is to develop an order N integration scheme for this propagation integral using a Gaussian quadrature.

One important thing to note is that Eqs (9) and (14) are exact solutions of the diffusion equation over a time step Δt . Therefore, Δt can be arbitrarily large. Importantly, it is not constrained by having to be small to numerically approximate a time derivative, as it must be in other methods like FTCS, BTCS, and Crank-Nicolson. Δt is then defined only by the time course of the flux source, the rate at which it turns on and off. This is one way that the propagation method provides an advantage over other existing alternatives. Moreover, because Δt can be anything, the propagation method is unconditionally stable.

Non-dimensionalization

The first step is non-dimensionalizing r with the diffusion lengthscale by defining

$$R = \frac{r}{2\sqrt{D\Delta t}} \quad (15)$$

with a similar definition for R' . One can then define

$$\rho(R, t_n) = R \cdot c(2\sqrt{D\Delta t}R, t_n) \quad (16)$$

(and similarly for $\rho_{\text{prop}}(R, t_n)$ and $\rho_{\text{source}}(R, t_n)$) so that the propagator integral (14) becomes

$$\rho_{\text{prop}}(R, t_{n+1}) = \int_0^\infty \rho(R', t_n) W(R, R') dR' \quad (17)$$

with the weight function

$$W(R, R') = \frac{2}{\sqrt{\pi}} \exp(-R^2) \exp(-(R')^2) \sinh(2RR') \quad (18)$$

$$= \frac{1}{\sqrt{\pi}} [\exp(-(R - R')^2) - \exp(-(R + R')^2)]. \quad (19)$$

For the source concentration,

$$\rho_{\text{source}}(R, t_{n+1}) = \chi_n F_n \text{erfc}(R) \quad (20)$$

where the flux factor F_n is defined as

$$F_n = \frac{j_n}{8\pi D^{3/2} \Delta t^{1/2}}. \quad (21)$$

Then,

$$\rho(R, t_{n+1}) = \rho_{\text{prop}}(R, t_{n+1}) + \rho_{\text{source}}(R, t_{n+1}). \quad (22)$$

Propagation algorithm

The integral in Eq (17) may be evaluated in a number of different ways (e.g., the Fast Gauss Transform [10]). The alternative approach taken here is to find a set of Gauss-diffusion quadrature (GDQ) points $\{x_\alpha\}_{\alpha=1}^N$ with corresponding weights $\{w_\alpha\}_{\alpha=1}^N$ that both depend on R . Then,

$$\rho(R, t_{n+1}) \approx \sum_{\alpha=1}^N \rho(x_\alpha(R), t_n) w_\alpha(R) + \chi_n F_n \operatorname{erfc}(R). \quad (23)$$

This, however, leads to an infinite nesting of grid point evaluations:

$$\rho(x_\beta, t_{n+1}) \approx \sum_{\alpha=1}^N \rho(x_\alpha(x_\beta), t_n) w_\alpha(x_\beta) + \chi_n F_n \operatorname{erfc}(x_\beta) \quad (24)$$

$$\rho(x_\alpha(x_\beta), t_n) \approx \sum_{\gamma=1}^N \rho(x_\gamma(x_\alpha(x_\beta)), t_{n-1}) w_\gamma(x_\alpha(x_\beta)) + \chi_{n-1} F_{n-1} \operatorname{erfc}(x_\alpha(x_\beta)) \quad (25)$$

$$\vdots \quad (26)$$

This is, of course, not desirable because we want a finite, well-defined set of GDQ points $\{x_\alpha\}$. One can overcome this problem by defining a set of grid points $\{R_i\}_{i=1}^I$ between which interpolation is used to evaluate the function at the GDQ points. Specifically, for each R_i , $\rho(R_i, t_{n+1})$ can be calculated from Eq (23) after the $\{x_\alpha(R_i)\}$ and $\{w_\alpha(R_i)\}$ have been calculated. The interpolation grid and the corresponding GDQ points and weights may be precalculated and saved.

The interpolation used here is a weighted sum of the interpolation points, so that

$$\rho(z) \approx \sum_{j=1}^I A_j^{(z)} \rho(R_j). \quad (27)$$

The interpolation weights depend on z because the R_j are nonuniformly spaced. By Eqs (17) and (23),

$$\rho_{\text{prop}}(R_i, t_{n+1}) \approx \sum_{\alpha=1}^N \rho(x_\alpha(R_i), t_n) w_\alpha(R_i) \quad (28)$$

$$\approx \sum_{\alpha=1}^N \sum_{j=1}^I A_j^{(x_\alpha(R_i))} \rho(R_j, t_n) w_\alpha(R_i) \quad (29)$$

$$= \sum_{j=1}^I W_{ij} \rho(R_j, t_n) \quad (30)$$

with the combined weights

$$W_{ij} = \sum_{\alpha=1}^N A_j^{(x_\alpha(R_i))} w_\alpha(R_i). \quad (31)$$

In order to express the iteration process in matrix/vector notation, define the column vector

$$\boldsymbol{\rho}_n = \{\rho(R_i, t_n)\}_{i=1}^I \quad (32)$$

and the matrix

$$W = \{W_{ij}\}_{i,j=1}^I. \quad (33)$$

Then,

$$\boldsymbol{\rho}_{n+1} = W\boldsymbol{\rho}_n + \chi_n F_n \mathbf{s} \quad (34)$$

for the source column vector

$$\mathbf{s} = \{\text{erfc}(R_i)\}_{i=1}^I. \quad (35)$$

One algorithm for evolving $\rho(R, t_n)$ in time then is:

1. Overhead (before the simulation)
 - a. Choose the parameters Δt , the maximum time T of the simulation, and the number of GDQ points N .
 - b. Calculate the interpolation grid $\{R_i\}$ for this T ; details are discussed in Subsection Interpolation Grid and Weights.
 - c. Compute the source vector $\mathbf{s}$ in [Eq \(35\)](#).
 - d. For each R_i , compute the GDQ points and weights $\{x_\alpha(R_i)\}_{\alpha=1}^N$ and $\{w_\alpha(R_i)\}_{\alpha=1}^N$; details are discussed in Subsection GDQ Points and [Appendix C](#).
 - e. For each i , compute the interpolation weights $\{A_i^{(x_\alpha(R_i))}\}_{\alpha=1}^N$; details are discussed in Subsection Interpolation Grid and Weights and [Appendix A](#).
 - f. Compute the combined weights matrix W using [Eq \(31\)](#).
2. (optional) Perform an error checking simulation; details are discussed in Section Error Analysis.
3. Simulation. Starting with $\rho(R, 0) = c_0 R$ for time $t = 0$,
 - a. evaluate the source state χ_n and flux j_n , possibly based on the concentrations $\boldsymbol{\rho}_n$ from the previous time step;
 - b. compute $\boldsymbol{\rho}_{n+1}$ from $\boldsymbol{\rho}_n$ using [Eq \(34\)](#).

The overhead is where all the serious programming work goes, specifically to compute the Gaussian quadrature, the interpolation grid, and the interpolation weights. That having been said, however, none of these steps are as difficult as they first appear to be. First, a *Mathematica* notebook that computes the Gaussian quadrature grid points and weights is available in the Supporting Information ([S1 File](#)). Second, programs like *Mathematica* provide function interpolation routines that can be used in the interpolation grid building strategy described in Interpolation Grid and Weights, thereby requiring little actual programming by the user. Lastly, Fornberg [11] provides an easy-to-implement algorithm for interpolation that, with one tweak, is very fast to compute and easily allows the user to change the interpolation order. The overhead calculations are generally fast, taking a few seconds and, if parallelization is used, less than 1 sec.

At each simulation step, the iteration (Eq (34)) requires a matrix/vector multiplication with the $I \times I$ matrix W . However, W is both small and sparse. Generally, W is small because $I < 200$ is generally sufficient for accurate results. This is because the interpolation grid is non-uniformly spaced and chosen based on knowledge of the exact solution of the diffusion problem, thereby *a priori* putting extra points where they are needed (Section Interpolation Grid and Weights). W is also sparse because only a small number of neighboring points are used in the Fornberg interpolation. In sample calculations, W ranged from 3% to 42% filled, depending on the number of GDQ and Fornberg points, as well as on the spacing of the interpolation grid points (specifically, the number of near and far points, as defined in Interpolation Grid and Weights). The sparsity of W and the small size of W make the propagation scheme very fast.

Interpolation grid and weights

There are two components to the interpolation used in the propagation algorithm, namely choosing the points $\{R_i\}$ and choosing an interpolation method. For the latter, we use a method by Fornberg [11] that uses polynomial interpolation between previously chosen points. This was chosen not only because it is easy to implement and quick to compute, but also because changing of the interpolation order N_F (i.e., the order of the interpolating polynomial or, equivalently, the number of nearest-neighbor points around each R_i) is effortless for the user. The technical details are given in Appendix A.

The interpolation grid points should be a minimal number of points, and therefore it would be best to have some knowledge of the mathematical structure of the solution $\rho(R, t_n)$. Luckily, it is possible to derive an exact solution (see Appendix B). Specifically,

$$\rho_n(R) = \sum_{m=0}^n \chi_{n-m} F_{n-m} e_m(R) \quad (36)$$

where

$$e_m(R) = \begin{cases} \operatorname{erfc}(R) & m = 0 \\ \operatorname{erf}\left(\frac{R}{\sqrt{m}}\right) - \operatorname{erf}\left(\frac{R}{\sqrt{m+1}}\right) & m \geq 1. \end{cases} \quad (37)$$

While one can use this exact solution, it is computationally impractical because no work from time t_n can be reused for time t_{n+1} . Therefore, if T timesteps have elapsed, it takes $O(T^2)$ operations per location R to compute $\rho(R, T)$ via Eq (36), which becomes slower than the propagation method proposed here after ~ 100 to 1000 timesteps.

Eq (36) does, however, provide useful information for making the interpolation grid. Specifically, one can find an interpolation grid for the functions $e_m(R)$ for various m and taking their union. This will yield points that are located where they are needed most.

The first step is to find the largest possible distance $R_{\max}$ that could be needed in the simulation that ends at time $T_{\max}$. This can be determined by considering the worst-case scenario where the current source is on the entire time. $R_{\max}$ should be chosen so that the concentration there is below some chosen threshold ε ; that is, $R_{\max}$ is the $r_{\max}$ that satisfies

$$\frac{j_{\max}}{4\pi D} \frac{1}{r_{\max}} \operatorname{erfc}\left(\frac{r_{\max}}{2\sqrt{D\Delta t}}\right) = \varepsilon \quad (38)$$

or, in nondimensionalized variables,

$$F_{\max} \operatorname{erfc}(R_{\max}) = \varepsilon R_{\max} \quad (39)$$

Fig 1. The functions $e_m(R)$ defined in Eq (37). The hash marks above the main figure are the interpolation grid points for small t (Simulation 1 in Table 1).

doi:10.1371/journal.pone.0132273.g001

where $F_{\max}$ corresponds to the largest possible flux $j_{\max}$ encountered during the simulation. The interpolation grid then spans $[0, R_{\max}]$.

The second step is to analyze the functions $e_m(R)$. For each $m \geq 1$, $e_m(0) = e_m(\infty) = 0$ and $e_m(R)$ has one maximum. Moreover, as $m \rightarrow \infty$, the maximum value decays to 0 and so the $e_m(R) \rightarrow 0$ for all R . This is shown in Fig 1. The figure shows that the first few $e_m(R)$ are the most important numerically and that their contribution is limited to the interval $[0, 10]$ (approximately). We therefore divide the interval $[0, R_{\max}]$ into two parts, the near ($R \leq 10$) and the far ($R > 10$) intervals. We start with the same uniform grid for each m , but the far interval is divided uniformly on a logarithmic scale while the near interval is divided on a linear scale. That is, $[0, 10]$ is divided uniformly while it is $[\log(10), \log(R_{\max})]$ that is divided uniformly. This serves two purposes. First, it focuses the points where $e_1(R)$ has the most structure (Fig 1) and needs to be resolved the best, namely in the near interval. Second, it keeps the number of grid points to a minimum. Since one can have $R_{\max} > 5000$, using the logarithmic scale places relatively few points in the far interval. More points (e.g., by linearly dividing the far interval) are not necessary because the $e_m(R)$ have little structure in the far interval that needs to be resolved (Fig 1). Also, since $R_{\max}$ increases with $T_{\max}$, the number of grid points grows logarithmically with $T_{\max}$.

After dividing the near and far intervals uniformly, the next step is to interpolate the functions $e_m(R)$ on these intervals separately. To do that, polynomial interpolation (usually third order) is used on each subinterval of the uniformly divided interval. Specifically, each subinterval is bisected and both e_m and the polynomial approximation are evaluated at this midpoint. Each interval is further bisected until e_m and the interpolation are within a specified tolerance. This is done for many m and the union of all these interpolation grids is taken. This does not add many points because each m starts with the same uniformly-spaced intervals that are then bisected. Some m (especially the small m) will have more bisection steps because they have

more structure, but this does not affect the accuracy of the m that do not have many bisections; in fact it only increases their accuracy. In practice, doing this for m up to 500 suffices, with significantly larger m adding only a few extra points. Fig 1 shows an example grid. It was generated using the *Mathematica* function `FunctionInterpolation`, which uses this interpolation scheme.

GDQ points

Gaussian quadrature integration methods are very efficient when numerically integrating a smooth function $f(R')$ against a weight function $\Omega(R')$ over integration limits a and b by using a weighted sum:

$$\int_a^b f(R')\Omega(R')dR' \approx \sum_{\alpha=1}^N f(R_{\alpha})\omega_{\alpha}. \quad (40)$$

The efficiency over standard integration methods (e.g., the trapezoidal rule) comes from being able to choose the locations R_{α} as well as the weights ω_{α} ; standard integration methods prescribe the R_{α} , usually as uniformly-spaced points. With the appropriate choices of N Gaussian weights and points, a polynomial of degree $2N$ is integrated exactly [12]. In practice, even a small N gives very accurate results if f is a very smooth function like we have here (Eq (36) and Fig 1). For the diffusion problem, N between 10 and 20 works well (discussed in detail later).

The theory of Gaussian quadratures is well-established (see, for example, Ref. [12]) and for many specific choices of Ω , a , and b there are standard techniques for computing the Gaussian weights and points for a given N . These are generally referred to as, for example, Gauss-Legendre ($\Omega = 1$, $a = -1$, $b = 1$) and Gauss-Laguerre ($\Omega = e^{-R'}$, $a = 0$, $b = \infty$) quadratures, and so in our case we refer to it as Gauss-diffusion quadrature, or GDQ. Computing a Gaussian quadrature for non-standard weight functions is possible [12], and Appendix C describes in detail how for the diffusion weight function of Eq (18). A *Mathematica* notebook that computes the GDQ weights and points is included in the Supporting Information (S1 File).

Error analysis

There are several different ways to analyze the accuracy of the propagation technique. By using a concrete example, we next show that it is possible to quantify the error in the simulations using a standardized system. Moreover, that system can also be used to find simulation parameters that achieve a desired accuracy with the shortest computation time.

Summary of error analysis

Knowing the accuracy of a simulation method is always important, but there are additional reasons for quantifying the error of the propagation method. First, by using interpolation we are introducing errors into the calculation of ρ_n at every time step and using those results to compute ρ_{n+1} . It is therefore a real concern that errors accumulate and lead to first-order errors after the many timesteps required for a simulation. Second, when the flux source is randomly on, $\rho_n(R)$ can have local maxima and minima that must be resolved accurately. If they are not, then that error may lead to spurious results at later times. With the example in the next section we show that neither of these occur when the simulation parameters are chosen well. For the vast majority of simulation parameter sets (i.e., N , N_F , I_{near} , and I_{far}) the propagation technique gives very accurate answers, although for some combinations of parameters the ρ_n can diverge. It is therefore important to run these kinds of test simulations.

There are two different checks one can do. The first is a simulation where the flux source is turned on and off randomly and the exact solution in Eq (36) is used to check the result. The second is a simulation where the flux source is always on with a constant flux and the exact solution in Eq (4) is used. Both have pros and cons. With the randomly-on source simulation only relatively small total simulation times T can be checked because the time to calculate the exact solution grows as T^2 , as discussed above. However, with this technique one can explicitly see whether the fine details of the $\rho_n(R)$ profile is reproduced. On the other hand, the advantage of the always-on simulation test is that simulations of arbitrary length can be checked to assure against error accumulation. In the next section, it is shown that the always-on test suffices because it bounds the error; its error is always larger than in the random-on test. One can therefore always run this test to assess simulation accuracy. Moreover, the propagation technique is fast enough that one can test which combination of simulation parameters gives a desired level of accuracy in the fastest time, thereby ensuring accurate results in minimal time. This is important for applications like calcium-induced calcium release where many similar simulations must be done.

Example

To illustrate this, we consider a specific example and examine the errors produced by different parameter choices. Specifically, the point source had a flux of 10^7 particles per second, which is of the order of the flux through an ion channel (approximately 1.6 pA of current for a univalent ion). The initial concentration c_0 was 0. The time step was $\Delta t = 10^{-7}$ seconds and the diffusion coefficient was 10^{-9} m/s², which gives $F_n = 6.61 \times 10^{-5}$ M for all time steps. A maximum of 10^6 time steps were considered for all simulations. Solving Eq (39) with $\varepsilon = 10^{-17}$ gives $R_{\max} = 5148$.

All the errors shown here are the maximum absolute difference between the simulated result ρ_{sim} and the exact result ρ_{exact} over all R , scaled by F_n :

$$\varepsilon_{\text{sim}}(t) = \frac{1}{F_n} \max_R |\rho_{\text{sim}}(R, t) - \rho_{\text{exact}}(R, t)|. \quad (41)$$

With this scaling, the exact result is independent of the flux and diffusion coefficient (see Eq (36)). Also, $\rho_{\text{exact}}(R, t)/F_n \leq 1$ for all R , and, because it is $O(1)$, $-\log_{10}(\varepsilon_{\text{sim}})$ is approximately the number of significant figures the simulation has correct. It is important to note, however, that this may not be the error metric of choice for a given problem. In this example, for instance, having a relatively large error with $\varepsilon_{\text{sim}} = 10^{-3}$ corresponds to a concentration error of 66.1 nM ($F_n \varepsilon_{\text{sim}}$), and if nothing in the problem is sensitive to nanomolar concentrations, then having a stricter error requirement like $\varepsilon_{\text{sim}} = 10^{-5}$ is overkill and probably a waste of computational time and effort.

As a first test, we consider different N (the number of Gaussian integration points) and N_F (the number of Fornberg interpolation points) using 173 interpolation grid points (80 near points for $R \leq 10$ and 73 far points for $R > 10$). The results for simulations lasting 10^4 time steps are shown in Fig 2. The flux source was always on. In general, as N increases, the error decreases. However, the same is not always true for N_F . Usually the error decreases for $N_F \leq 8$, but for larger N_F the simulation ρ can diverge (e.g., values $> 10^{10}$). This is not unexpected, however, as high interpolation order does not necessarily produce high interpolation accuracy. For example, a constant function is well-approximated by linear interpolation ($N_F = 1$), but poorly by high-order polynomials because these necessarily require oscillations between the grid points in order to be 0 at the grid points. This is, in fact, what occurs here; the interpolation oscillates far from the flux source where ρ is constant (at 0). As the simulation continues, these

Fig 2. Error for different N_F (x-axis) and N (y-axis) for simulations lasting 10^4 time steps. The flux source was always on. The maximum difference between the simulated ρ and exact solution at all R over all 10^4 time steps ($\max_t\{\varepsilon_{\text{sim}}(t)\}$) is shown on a color-coded $\log_{10}$ scale. Errors larger than 1 were truncated to 1 to make the graph more readable.

doi:10.1371/journal.pone.0132273.g002

oscillations are either damped out for the best results (purple in Fig 2) or amplified for the diverged results (red) (data not shown). If divergence does occur, the user should be aware that small changes in N and N_F can alleviate this problem (Fig 2).

With $N = 20$ and $N_F = 9$ that produce accurate results, as a second test we consider the number of interpolation grid points I . More specifically, we consider how the number of near points I_{near} (for $0 \leq R \leq 10$, as discussed in Interpolation Grid and Weights) and the number of far points I_{far} ($R > 10$) affect the error. Table 1 lists the six cases that were considered, and Fig 3 shows $\varepsilon_{\text{sim}}(t)$ for the first five cases; the results of Simulation 6 were identical to Simulation 5 except that in Fig 3A it did not have the upswing in error near the 10^6 -th time step that occurs in Simulation 5 (and overlaps with Simulation 2).

Table 1. Parameters used in Fig 3. The circled numbers in Fig 3 correspond to the simulation number (Sim.) in the Table. For each simulation, the number of near (I_{near}) and far (I_{far}) interpolation grid points are shown, as well as the sum (I). Simulation 6 is not shown in Fig 3 because it was the same as Simulation 5 except that it did not have the uptick in error at the end of the simulation. For all simulations, $N = 20$ and $N_F = 9$.

Sim.	I_{near}	I_{far}	I
1	19	45	64
2	19	90	109
3	19	180	199
4	83	51	134
5	83	90	173
6	83	180	263

doi:10.1371/journal.pone.0132273.t001

Fig 3. Error versus simulation time for five of the six simulations listed in Table 1. The circled numbers correspond to the Simulation listed in the table. (A) The flux source is always on. (B) The flux source is randomly on for 50000 time steps. The bars on the right side are the maximum error over 50000 time steps from panel A to show that the error in the always-on simulations bounds the error of the randomly-on simulations. For all simulations, $N = 20$ and $N_F = 9$.

doi:10.1371/journal.pone.0132273.g003

We first analyze the case where the flux source is on at all times (Fig 3A). To make sense of the results, consider Simulations 1, 2, and 3, which differ in the number of far points. At the beginning of the simulation, they overlap. At later times, they separate and eventually Simulations 1 and 2 begin to diverge. This indicates that increasing number of far points increases accuracy at later times. This is consistent with the results of Simulations 4, 5, and 6. Again, they overlap at early times and separate according to how many far points there are; the more far points, the smaller the late-time error. Conversely, the more near points, the smaller the early-time error. For example, compare Simulations 1 and 4 and Simulations 2 and 5. Intuitively, these results make sense: early in the simulation ρ is changing near the source so the near interval is most important, while late in the simulation ρ is changing in the far interval.

In the case just considered, the flux source is always on. However, the propagation technique is designed for fluctuating sources. Moreover, the always-on case does not require the specially-constructed interpolation grid. This is shown in Fig 4 where ρ profiles at different times are shown for a randomly-on flux source. For comparison, the dashed line shows the always-on result at the same time step as the red solid curve. The random-on profiles can have multiple local maxima and inflection points, something not seen for the monotonically decaying always-on profile. Because of the more complicated ρ profiles, it is possible that the error for the randomly-on case is much different than for the always-on case. For example, not resolving those maxima or minima at one time can propagate an error to later times.

Fig 3B shows that this is not the case. In fact, for all six cases considered, the maximum error is always less for the random-on case than for the always-on case. This is very useful because the exact answer is much easier to compute for the always-on case; for the random-on case only about 5×10^5 time steps can be computed in a reasonable amount of time. Bounding the error is also important because it makes it always possible to pre-compute the error; if the details of how the flux source turns on or off are unimportant for measuring simulation error, then one can use the always-on case.

Fig 4. $\rho(R)$ for a randomly-on flux source at various simulation times. It illustrates both the complicated structure of these curves and to show how the interpolation grid points are more dense in the regions that need them. The points are the simulation results and the curves are the exact result. For comparison, the dashed curve is the always-on exact result at the same time step as the red curve. For clarity a very small number of interpolation grid points were used ($l = 64$).

doi:10.1371/journal.pone.0132273.g004

Moreover, one can optimize accuracy versus computation speed by combining all of these error checks. The computation speed of the propagation method is dominated by the matrix/vector multiplication in Eq (34), and because W is sparse, the number of multiplications required is the number of nonzero entries in W . Therefore, one can vary all the parameters above (N , N_F , I_{near} , and I_{far}) and plot $\max_t \{\epsilon_{\text{sim}}(t)\}$ versus the number of nonzero entries in W . The fastest and most accurate set of parameters can then be chosen for all future simulations. Fig 5 shows this for 336 parameter choices for 10^6 time steps. The arrow points to the parameter set with the fastest computation time that still achieve high accuracy ($\max_t \{\epsilon_{\text{sim}}(t)\} \leq 10^{-5}$), in this case $N = 10$, $N_F = 8$, $I_{\text{near}} = 25$, and $I_{\text{far}} = 90$ with W having 1356 nonzero entries. Because the propagation method is fast, it is possible to scan this large set of parameters; Fig 5 took approximately 7 hours on a desktop computer with a 6-core 3.33 GHz i7-980X processor using *Mathematica* version 9 (Wolfram Research, Champaign, Illinois), which could have been significantly faster if done in parallel rather than in series. This time investment is well worth it for applications where a large number of simulations must be done because it guarantees both high accuracy and the fastest overall computation speed for production runs.

To see how this compares to classical methods like BTCS and Crank-Nicolson, one must first consider their mathematical structure. Each discretizes both the time and space derivatives in Eq (1), almost always with uniform spacing Δt and Δr for the time and space coordinates, respectively. This creates a system of linear equations to be solved at each time step with a tridiagonal matrix [12]. Since tridiagonal systems can be solved with $O(N_r)$ operations (N_r is the number of spatial grid points), to be comparable to the propagation method, N_r should be around 1000.

N_r is determined by both Δr and the maximum distance L to be considered. Since the diffusion equation requires both initial and boundary conditions at $r = 0$ and $r = L$, one must pick L large enough so that the concentration is (approximately) 0. For the example in this section, $R_{\text{max}} = 5148$ corresponds to $L = 102.96 \mu\text{m}$. From Fig 4 one can see that it takes a resolution in R of at least 0.25, which corresponds to 5 nm for the example in this section. Therefore, $N_r \equiv$

Fig 5. Simulation error versus the number of nonzero elements in W (to represent computation time). Each point is for a different parameter set (N , N_F , l_{near} , and l_{far}) for a long simulation of 10^6 time steps for the example described in the main text with the flux source always on. The arrow points to the optimal balance of high accuracy and computation speed. Errors larger than 1 were truncated to 1 to make the graph more readable.

doi:10.1371/journal.pone.0132273.g005

$[L/\Delta r] = 20592$. Even for such a large system, a simple Crank-Nicolson implementation produced $\max_t \{\epsilon_{\text{sim}}(t)\} > -4.1$, significantly worse than many propagation implementations with suboptimal parameters (Fig 5) and 6 to 45 times as many nonzero matrix entries.

Extension to reaction-diffusion

The focus so far has been on the diffusion equation with a point source (Eq (1)). However, the work done so far can also be extended to include a simple model where chemical reactions remove the diffusing particles. If we assume that these buffer molecules are present at high concentrations and do not change in time or space, then Eq (1) becomes [13]

$$\frac{\partial c(r, t)}{\partial t} = \frac{D}{r^2} \frac{\partial}{\partial r} \left(r^2 \frac{\partial c}{\partial r}(r, t) \right) + s(r, t) + k_- B - k_+ b c(r, t) \quad (42)$$

where k_- is the off-rate of the buffer and k_+ is the on-rate. Since we assume the buffer is present at high concentration, the free buffer concentration b and the bound buffer concentration B are known constants (see, for example, [13]). Eq (6) then has two new source and sink terms whose form is the same as Eq (8). For the source term from the diffusant unbinding from the buffer,

$$c_-(r, t_{n+1}) = k_- B \int_{t_n}^{t_{n+1}} \int_{\mathbb{R}^3} (4\pi D(t_{n+1} - t'))^{-3/2} \exp \left(-\frac{|\mathbf{r} - \mathbf{r}'|^2}{4D(t_{n+1} - t')} \right) d\mathbf{r}' dt' \quad (43)$$

$$= k_- B \int_{t_n}^{t_{n+1}} \frac{dt'}{\sqrt{D(t_{n+1} - t')}} \quad (44)$$

$$= 2k_- B \sqrt{\frac{\Delta t}{D}} \quad (45)$$

where Eq (14) with a constant concentration was used to obtain the intermediate equation. For the sink term from the diffusant binding to buffer,

$$c_+(r, t_{n+1}) = k_+ b \int_{t_n}^{t_{n+1}} \int_{\mathbb{R}^3} (4\pi D(t_{n+1} - t'))^{-3/2} \exp\left(-\frac{|\mathbf{r} - \mathbf{r}'|^2}{4D(t_{n+1} - t')}\right) c(r', t') d\mathbf{r}' dt' \quad (46)$$

$$\approx k_+ b \Delta t \int_{\mathbb{R}^3} (4\pi D \Delta t)^{-3/2} \exp\left(-\frac{|\mathbf{r} - \mathbf{r}'|^2}{4D \Delta t}\right) c(r', t_n) d\mathbf{r}' \quad (47)$$

$$= k_+ b \Delta t \cdot c_{\text{prop}}(r, t_{n+1}) \quad (48)$$

where the intermediate equation used a one-point quadrature for the time integral and the final result follows from Eq (7).

All of the results of the previous sections then carry forward directly:

$$\rho(R, t_{n+1}) = (1 - k_+ b \Delta t) \rho_{\text{prop}}(R, t_{n+1}) + \rho_{\text{source}}(R, t_{n+1}) + 4k_- B \Delta t \cdot R \quad (49)$$

and Eq (34) becomes

$$\boldsymbol{\rho}_{n+1} = (1 - k_+ b \Delta t) W \boldsymbol{\rho}_n + \mathbf{s}'_n \quad (50)$$

where

$$\mathbf{s}'_n = \chi_n F_n \mathbf{s} + 4k_- B \Delta t \mathbf{R} \quad (51)$$

with

$$\mathbf{R} = \{R_i\}_{i=1}^I. \quad (52)$$

Conclusion

An integration quadrature for the propagation integral of the spherically-symmetric diffusion equation from a time-dependent point source was developed. This integral (Eq (17)) was evaluated using a new Gauss-diffusion quadrature and a specialized interpolation grid was used to find the values at the Gaussian quadrature points for the next integration. This scheme then balances accuracy with speed by using a small number of integration and interpolation grid points to achieve high accuracy over many time steps.

The analysis of this propagation technique shows that it has a number of positive attributes:

1. It works with a small number of grid points. Even with just 115 interpolation grid points one can get very accurate results even for long simulations (e.g., 10^6 time steps in Fig 5) with a propagation matrix with < 1400 nonzero entries.
2. The update step is a simple sparse-matrix/vector multiplication.
3. The simulation error is quantifiable before any “real” simulations are done.
4. Simulation error and computation time can be optimized together to minimize both error and simulation time.
5. The time step Δt can be arbitrarily large, being defined only by the rate at which the flux source turns on and off.

6. It is unconditionally stable.
7. It can easily incorporate nonlinear feedback into the flux source, for example, from the concentration profile calculated at previous time steps.

The algorithm developed here is an alternative to more traditional finite difference or finite element approaches. It is different since it computes the concentration profile at the next time step using a simple sparse-matrix/vector multiplication rather than a (sparse) matrix solve needed for unconditionally-stable implicit-time methods. Differences in computation speed then depend on the number of grid points used in the traditional approaches and the specifics of the matrix solution method, making it difficult to compare the methods head-to-head. However, if uniform spatial discretization is used in classical methods like BTCS or Crank-Nicolson, they produce matrices with significantly more nonzero entries than the propagation method. The matrix/vector multiplication of the propagation method is similar to that of conditionally-stable explicit-time methods, except that these methods require small time steps to be convergent, while the time step for the propagation method is solely determined by the timescale on which the flux source turns on and off. However, the propagation method is meant to be an alternative technique that tries to minimize the number of points (and nonzero matrix entries) while retaining high accuracy.

Appendix A: Revised Fornberg algorithm

So that the paper is self-contained, a brief summary of Fornberg's interpolation algorithm [11] is given, noting one time-saving change and focusing only on function interpolation, rather than derivatives as well; that is, here $M = 0$ in Fornberg's notation. If there are grid points $\{R_i\}_{i=0}^{I-1}$ to interpolate from, then for some given point x_0 we seek an approximation of $f(x_0)$ as a weighted average of $N_F \leq I$ of these interpolation points:

$$f(x_0) \approx \sum_{v=0}^{N_F-1} A_v^{(x_0)} f(R_v^*) \quad (53)$$

where the $\{R_v^*\}_{v=0}^{N_F-1}$ is the subset of N_F interpolation points closest to x_0 . The superscript (x_0) indicates that the weights A_v are different for different x_0 .

It is important to note that the Fornberg algorithm uses only the first N_F number of interpolation points given to it. Therefore, for each x_0 one must input the N_F interpolation points nearest to x_0 . In particular, for each x_0 a different set of interpolation points must be given; if one always inputs $\{R_i\}$, then the first N_F of them will always be used to compute $f(x_0)$, even if x_0 is not close to any of these points. Hence the R_v^* notation in Eq (53). This is a point not made clearly by Fornberg.

The Fornberg algorithm for function interpolation only (i.e., not evaluating derivatives) is:

```

set all  $a_{k,l} = 0$  for  $0 \leq k, l \leq N_F - 1$  and  $a_{0,0} = 1$ 
set  $c_1 = 1$ 
for  $n = 1$  to  $N_F - 1$  (this is the time-saving change; Fornberg has  $I$  instead of  $N_F - 1$  which results in an extremely long calculation of many 0's)
    set  $c_2 = 1$ 
    for  $v = 1$  to  $N_F - 2$ 
        set  $c_3 = R_n - R_v$ 
        set  $c_2 = c_2 c_3$ 
        set  $a_{n,v} = (R_n - x_0) a_{n-1,v} / c_3$ 

```

```

next v
set  $a_{n,n} = -\frac{c_1}{c_2}(R_{n-1} - x_0)a_{n-1,n-1}$ 
set  $c_1 = c_2$ 
next n
set  $A_v^{(x_0)} = a_{N_F-1,v}$ 

```

Appendix B: Derivation of the exact solution

Here, a brief derivation of the exact solution to the propagation integral evolution is given, specifically that

$$\rho_n(R) = \sum_{m=1}^n \chi_{n-m} F_{n-m} \left[\operatorname{erf}\left(\frac{R}{\sqrt{m}}\right) - \operatorname{erf}\left(\frac{R}{\sqrt{m+1}}\right) \right] + \chi_n F_n E(R) \quad (54)$$

where

$$E(R) = \operatorname{erfc}(R). \quad (55)$$

While this result is almost surely not new, it was derived independently by the author in the following way.

Define

$$\bar{\rho}_n(R) = \begin{cases} \rho_n(R) & \text{if } R \geq 0 \\ -\rho_n(-R) & \text{if } R < 0 \end{cases} \quad (56)$$

and note that the weight function of [Eq \(19\)](#) has the property

$$W(R, -R') = W(-R, R') = -W(R, R'). \quad (57)$$

Then

$$\bar{\rho}_n(R) = \frac{1}{2} \int_{-\infty}^{\infty} \bar{\rho}_{n-1}(R') W(R, R') dR' + \chi_n F_n E(R). \quad (58)$$

Taking the Fourier transform (denoted with $\tilde{\cdot}$ and $\mathcal{F}_R$ to be explicit of the variable R), we get

$$\tilde{\bar{\rho}}_n(\omega) = \frac{1}{2} \int_{-\infty}^{\infty} \bar{\rho}_{n-1}(R') \mathcal{F}_R(W(R, R')) dR' + \chi_n F_n \tilde{E}(\omega) \quad (59)$$

with

$$\mathcal{F}_R(W(R, R')) = \frac{1}{\sqrt{2\pi}} e^{-\omega^2/4} (e^{i\omega R'} - e^{-i\omega R'}). \quad (60)$$

Therefore, with $*$ denoting the complex conjugate,

$$\tilde{\bar{\rho}}_n(\omega) = \frac{1}{2} e^{-\omega^2/4} (\tilde{\bar{\rho}}_{n-1}(\omega) - \tilde{\bar{\rho}}_{n-1}(\omega)^*) + \chi_n F_n \tilde{E}(\omega) \quad (61)$$

$$= i e^{-\omega^2/4} \operatorname{Im}(\tilde{\bar{\rho}}_{n-1}(\omega)) + \chi_n F_n \tilde{E}(\omega). \quad (62)$$

Assuming the initial concentration is 0, we have

$$\rho_0(R) = \chi_0 F_0 E(R) \quad (63)$$

and

$$\tilde{\rho}_0(\omega) = \chi_0 F_0 f_0(\omega) \quad (64)$$

where

$$f_0(\omega) = \sqrt{\frac{2}{\pi}} \frac{1 - e^{-\omega^2/4}}{\omega}. \quad (65)$$

Using [Eq \(62\)](#) repeatedly, we find that

$$\text{Im}(\tilde{\rho}_n(\omega)) = \sum_{m=0}^n \chi_{n-m} F_{n-m} e^{-m\omega^2/4} f_0(\omega). \quad (66)$$

Using the fact that

$$\mathcal{F}^{-1}(ie^{-m\omega^2/4} f_0(\omega)) = \text{erf}\left(\frac{R}{\sqrt{m}}\right) - \text{erf}\left(\frac{R}{\sqrt{m+1}}\right), \quad (67)$$

taking the inverse Fourier transform of [Eq \(62\)](#) gives that

$$\rho_n(R) = \mathcal{F}^{-1}(ie^{-\omega^2/4} \text{Im}(\tilde{\rho}_{n-1}(\omega))) + \chi_n F_n E(R) \quad (68)$$

$$= \sum_{m=0}^{n-1} \chi_{n-1-m} F_{n-1-m} \mathcal{F}^{-1}(ie^{-(m+1)\omega^2/4} f_0(\omega)) + \chi_n F_n E(R) \quad (69)$$

$$= \sum_{m=1}^n \chi_{n-m} F_{n-m} \left[\text{erf}\left(\frac{R}{\sqrt{m}}\right) - \text{erf}\left(\frac{R}{\sqrt{m+1}}\right) \right] + \chi_n F_n E(R) \quad (70)$$

which is the same as [Eq \(36\)](#).

Appendix C: Gaussian quadrature

The theory of Gaussian quadratures is developed in a number of texts (e.g., that by Press et al. [\[12\]](#)) so only a brief outline relevant to the diffusion problem is given. One must compute the coefficients a_j ($j = 0, 1, \dots, N$) and b_j ($j = 1, \dots, N$) for the orthonormal polynomials

$$p_{-1}(x) = 0 \quad (71)$$

$$p_0(x) = 1 \quad (72)$$

$$p_{j+1}(x) = (x - a_j)p_j(x) - b_j p_{j-1}(x) \quad (73)$$

where

$$a_j(R) = \frac{\langle x p_j(x)^2 \rangle}{\langle p_j(x)^2 \rangle} \quad (74)$$

and

$$b_j(R) = \frac{\langle p_j(x)^2 \rangle}{\langle p_{j-1}(x)^2 \rangle} \quad (75)$$

for the inner product

$$\langle f \rangle = \int_0^\infty f(x) W(R, x) dx. \quad (76)$$

Note that because the integral's weight function depends on R , the coefficients depend on R as well.

To determine the coefficients a_j and b_j , we need the moments of the weight function:

$$M_n \equiv \int_0^\infty (R')^n \cdot W(R, R') dR' \quad (77)$$

$$= \frac{2}{\sqrt{\pi}} R \exp(-R^2) \Gamma\left(\frac{n}{2} + 1\right) \Phi\left(\frac{n}{2} + 1, \frac{3}{2}; R^2\right) \quad (78)$$

where $\Phi(\alpha, \gamma; z)$ is the confluent hypergeometric function which is equal to the generalized hypergeometric series ${}_1F_1(\alpha; \gamma; z)$ [14]. This function has the recurrence relationship [14]

$$\Phi(\alpha + 1, \gamma; z) = \frac{z + 2\alpha - \gamma}{\alpha} \Phi(\alpha, \gamma; z) + \frac{\gamma - \alpha}{\alpha} \Phi(\alpha - 1, \gamma; z) \quad (79)$$

that will allow us to evaluate the moments efficiently. From the structure of this recurrence relationship, it is easiest to break the moments into even n and odd n .

For even $n = 2k$ ($k \geq 1$) with $\alpha = n/2 = k$, we have

$$\Phi\left(k + 1, \frac{3}{2}; R^2\right) = \frac{R^2 + 2k - \frac{3}{2}}{k} \Phi\left(k, \frac{3}{2}; R^2\right) + \frac{\frac{3}{2} - k}{k} \Phi\left(k - 1, \frac{3}{2}; R^2\right) \quad (80)$$

or, when

$$\phi_k = \Phi\left(k, \frac{3}{2}; R^2\right), \quad (81)$$

$$\phi_{k+1} = \frac{R^2 + 2k - \frac{3}{2}}{k} \phi_k + \frac{\frac{3}{2} - k}{k} \phi_{k-1}. \quad (82)$$

This gives a procedure for increasing k by starting with

$$\phi_0 = 1 \quad (83)$$

and

$$\phi_1 = \frac{\sqrt{\pi}}{2} \frac{\exp(R^2)}{R} \operatorname{erf}(R). \quad (84)$$

Using that

$$\Gamma\left(\frac{n}{2} + 1\right) = \Gamma(k + 1) = k!, \quad (85)$$

we get

$$M_0 = \frac{2}{\sqrt{\pi}} R \exp(-R^2) \cdot \phi_1 \quad (86)$$

$$= \operatorname{erf}(R) \quad (87)$$

and

$$M_2 = \frac{2}{\sqrt{\pi}} R \exp(-R^2) \cdot \phi_2 \quad (88)$$

$$= \frac{2}{\sqrt{\pi}} R \exp(-R^2) \cdot \left(\left(R^2 + \frac{1}{2} \right) \phi_1 + \frac{1}{2} \right) \quad (89)$$

$$= \left(R^2 + \frac{1}{2} \right) M_0 + \frac{1}{\sqrt{\pi}} R \exp(-R^2) \quad (90)$$

$$= \left(R^2 + \frac{1}{2} \right) \operatorname{erf}(R) + \frac{1}{\sqrt{\pi}} R \exp(-R^2), \quad (91)$$

so that for even $n \geq 4$

$$M_{2k} = \frac{2}{\sqrt{\pi}} R \exp(-R^2) \cdot k! \phi_{k+1} \quad (92)$$

$$= \frac{2}{\sqrt{\pi}} R \exp(-R^2) \cdot k! \left(\frac{R^2 + 2k - \frac{3}{2}}{k} \phi_k + \frac{\frac{3}{2} - k}{k} \phi_{k-1} \right) \quad (93)$$

$$= \frac{2}{\sqrt{\pi}} R \exp(-R^2) \cdot (k-1)! \left(\left(R^2 + 2k - \frac{3}{2} \right) \phi_k + \left(\frac{3}{2} - k \right) \phi_{k-1} \right) \quad (94)$$

$$= \left(R^2 + 2k - \frac{3}{2} \right) \cdot M_{2(k-1)} + (k-1) \left(\frac{3}{2} - k \right) \cdot M_{2(k-2)}. \quad (95)$$

Here [Eq \(95\)](#) is derived by substituting in [Eq \(92\)](#).

For odd $n = 2l + 1$ ($l \geq 1$) with $\alpha = l + 1/2$, we have

$$\Phi\left(l + \frac{3}{2}, \frac{3}{2}; R^2\right) = \frac{R^2 + 2l - \frac{1}{2}}{l + \frac{1}{2}} \Phi\left(l + \frac{1}{2}, \frac{3}{2}; R^2\right) + \frac{1-l}{l + \frac{1}{2}} \Phi\left(l - \frac{1}{2}, \frac{3}{2}; R^2\right) \quad (96)$$

so that

$$\phi_{l+3/2} = \frac{R^2 + 2l - \frac{1}{2}}{l + \frac{1}{2}} \phi_{l+1/2} + \frac{1-l}{l + \frac{1}{2}} \phi_{l-1/2}. \quad (97)$$

This gives a procedure for increasing l by using

$$\Gamma\left(\frac{n}{2} + 1\right) = \Gamma\left(l + 1 + \frac{1}{2}\right) = \frac{\sqrt{\pi}}{2^{l+1}} (2l+1)!! \quad (98)$$

and starting with

$$M_1 = \frac{2}{\sqrt{\pi}} R \exp(-R^2) \cdot \frac{\sqrt{\pi}}{2} \phi_{3/2} \quad (99)$$

$$= R \quad (100)$$

and

$$M_3 = \frac{2}{\sqrt{\pi}} R \exp(-R^2) \cdot \frac{3\sqrt{\pi}}{4} \Phi\left(\frac{5}{2}, \frac{3}{2}; R^2\right) \quad (101)$$

$$= \frac{3}{2} R \left(\frac{2}{3} R^2 + 1 \right) \quad (102)$$

$$= \frac{1}{2} R (2R^2 + 3) \quad (103)$$

so that

$$M_{2l+1} = \frac{2}{\sqrt{\pi}} R \exp(-R^2) \cdot \frac{\sqrt{\pi}}{2^{l+1}} (2l+1)!! \cdot \phi_{l+3/2} \quad (104)$$

$$= R \exp(-R^2) \cdot \frac{1}{2^l} (2l+1)!! \left(\frac{R^2 + 2l - \frac{1}{2}}{l + \frac{1}{2}} \phi_{l+1/2} + \frac{1-l}{l + \frac{1}{2}} \phi_{l-1/2} \right) \quad (105)$$

$$= \frac{2^{l-1} (2l+1)!!}{2^l (2l-1)!!} \frac{R^2 + 2l - \frac{1}{2}}{l + \frac{1}{2}} M_{2l-1} + \frac{2^{l-2} (2l+1)!!}{2^l (2l-3)!!} \frac{1-l}{l + \frac{1}{2}} M_{2l-3} \quad (106)$$

$$= \frac{1}{2} (2l+1) \frac{R^2 + 2l - \frac{1}{2}}{l + \frac{1}{2}} M_{2l-1} + \frac{1}{4} (4l^2 - 1) \frac{1-l}{l + \frac{1}{2}} M_{2l-3} \quad (107)$$

$$= \left(R^2 + 2l - \frac{1}{2} \right) M_{2(l-1)+1} - \left(l - \frac{1}{2} \right) (l-1) M_{2(l-2)+1}. \quad (108)$$

Here, [Eq \(107\)](#) is derived by substituting in [Eq \(104\)](#).

These moments can then be used to build a Gaussian quadrature using standard procedures, namely finding the eigenvalues $\{\lambda_j\}$ and eigenvectors $\{\mathbf{v}_j\}$ of the symmetric tridiagonal matrix [\[12\]](#)

$$\begin{pmatrix} a_0 & \sqrt{b_1} & 0 & \cdots & 0 \\ \sqrt{b_1} & a_1 & \sqrt{b_2} & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \sqrt{b_{N-2}} & a_{N-2} & \sqrt{b_{N-1}} \\ 0 & \cdots & 0 & \sqrt{b_{N-1}} & a_{N-1} \end{pmatrix}. \quad (109)$$

Then, the Gaussian grid points are the eigenvalues (i.e., $x_\alpha = \lambda_\alpha$) and the weights are

$$w_\alpha = v_{\alpha,0}^2 \cdot \int_0^\infty W(R, R') dR' \quad (110)$$

$$= v_{\alpha,0}^2 \cdot \text{erf}(R) \quad (111)$$

where $v_{\alpha,0}$ is the first element of $\mathbf{v}_\alpha$.

From a theoretical point of view this is straightforward. From a computational point of view, however, it is important to note that the inner product ratios that define the coefficients a_j and b_j in Eqs (74) and (75) must be done with extremely high precision (e.g., 100 digits or more of working precision) or round-off errors accumulate quickly, even for relatively small N . Moreover, the matrix in Eq (109) is extreme ill-conditioned. With similarly high working precision, however, finding the eigenvalues and eigenvectors is fast and effective. Lastly it is noted that computing the inner products,

$$\langle x^a p_j(x)^2 \rangle \equiv \int_0^\infty W(R, x) x^a \left(\sum_n c_n^{(j)} x^n \right)^2 dx \quad (112)$$

($a = 0$ or 1) can be done by working only with the coefficients $c_n^{(j)}$ of the polynomial p_j to find the coefficients of $p_j(x)^2$ by using a list convolution and then using a dot product with the vector of moments M_n .

This procedure gives GDQ points $\{x_\alpha(R)\}_{\alpha=1}^N$ and weights $\{w_\alpha(R)\}_{\alpha=1}^N$. Note that while the matrix and inner product operations must be done with many digits of accuracy, once the GDQ points and weights are calculated they can be truncated to machine precision for the diffusion calculations. A *Mathematica* notebook that computes the GDQ points and weights is available in the Supporting Information (S1 File).

An alternative approach to calculating the GDQ points and weights is the procedure described by Golub and Welsch [15], but ill-conditioning will still be an issue.

Supporting Information

S1 File. Mathematica notebook containing code to compute the GDQ points and weights as described in Appendix C.

(NB)

Acknowledgments

I would like to thank Matt Knepley and Jay Bardhan for very valuable discussions.

Author Contributions

Conceived and designed the experiments: DG. Performed the experiments: DG. Analyzed the data: DG. Wrote the paper: DG.

References

1. Fick A. Ueber diffusion. Ann der Physik. 1855; 94: 59 (1855:).
2. Fick A. On liquid diffusion. Phil Mag J Sci. 1855; 10: 30 (1855:).
3. Barton G. Elements of Green's functions and propagation. Oxford: Clarendon Press; 1989.
4. Crank J. The mathematics of diffusion. Oxford: Oxford University Press; 1980.
5. Cussler EL. Diffusion: Mass transfer in fluid systems. Cambridge: Cambridge University Press; 2009.

6. Greengard L, Strain J. The fast Gauss transform. *SIAM Journal on Scientific and Statistical Computing*. 1991; 12: 79–94. doi: [10.1137/0912004](https://doi.org/10.1137/0912004)
7. Stern MD, Ríos E, Maltsev VA. Life and death of a cardiac calcium spark. *J Gen Physiol* 2013; 142: 257–274. doi: [10.1085/jgp.201311034](https://doi.org/10.1085/jgp.201311034) PMID: [23980195](https://pubmed.ncbi.nlm.nih.gov/23980195/)
8. Conlisk AT. *Essentials of micro- and nanofluidics: With applications to the biological and chemical sciences*. Cambridge: Cambridge University Press; 2012.
9. Bers DM. *Excitation-contraction coupling and cardiac contractile force*. 2nd ed. Dordrecht: Kluwer Academic Publishers; 2001.
10. Li JR, Greengard L. On the numerical solution of the heat equation I: Fast solvers in free space. *J Comp Physics*. 2007; 226: 1891–1901. doi: [10.1016/j.jcp.2007.06.021](https://doi.org/10.1016/j.jcp.2007.06.021)
11. Fornberg B. Generation of finite difference formulas on arbitrary spaced grid. *Math Comp*. 1988; 51: 699–706. doi: [10.1090/S0025-5718-1988-0935077-0](https://doi.org/10.1090/S0025-5718-1988-0935077-0)
12. Press WH, Teukolsky SA, Vetterling W, Flannery BP. *Numerical Recipes in C*. Cambridge, UK: Cambridge University Press; 1992.
13. Neher E. Concentration profiles of intracellular calcium in the presence of a diffusible chelator. In: Heinemann U, Klee M, Neher E, editors. *Calcium Electrogenesis and Neuronal Functioning*. Berlin: Springer-Verlag. pp. 80–96; 1986.
14. Gradshteyn IS, Ryzhik IM. *Table of integrals, series, and products*. San Diego: Academic Press; 2000.
15. Golub GH, Welsch JH. Calculation of Gaussian quadrature rules. *Math Comp*. 1969; 23: 221–230. doi: [10.1090/S0025-5718-69-99647-1](https://doi.org/10.1090/S0025-5718-69-99647-1)