

# Discussing the use of reaction diffusion equations for the simulation of physical phenomena

Simon Abrelat

002129–0004

Math HL IA

May 2019

Words:

## **Abstract**

This is the thing working good (Turing, 1952), here is another example  
(P.Gray, 1984)

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Calculus</b>	<b>3</b>
<b>3</b>	<b>Diffusion</b>	<b>9</b>
<b>4</b>	<b>Reaction</b>	<b>11</b>
<b>5</b>	<b>Reaction Diffusion</b>	<b>12</b>
<b>6</b>	<b>Appendix</b>	<b>14</b>

# 1 Introduction

## 2 Calculus

Reaction diffusion relies on multi-variable calculus, from the partial derivative to the Laplacian operator that governs diffusion. Multi is a natural extension of calculus where instead of focusing on the accumulation and change on 1 axis, that being  $y$ , it is possible to do in to any number of dimensions. This means that if your function has more than 1 input, like for 3D functions, you would have a new arsenal of tools and methods to analyse your equations. The most simple being the limit which fundamentally operates the same as in the normal case. If there is a hole in the function but otherwise is continuous, it would “fill in” that hole, and if there is a jump discontinuity the limit is defined some certain directions but the limit itself is undefined. Once again using the concept of limits we can get the integral and derivative as in normal calculus; however for this application we will be primarily focusing on partial derivatives and other related operations. Instead derivatives in in multivariate partial derivatives are used. These are

computed in a similar form to derivatives but the input that is not differentiated is treated as a constant. For use in the reaction equations the partial differential equations behave in a similar way to differential equations but rely on more inputs but are still the rate of change in a given input over a function.

$$\frac{\partial f}{\partial x} = \partial_x f = \lim_{h \rightarrow 0} \frac{f(x+h, y) - f(x, y)}{h} \quad (1)$$

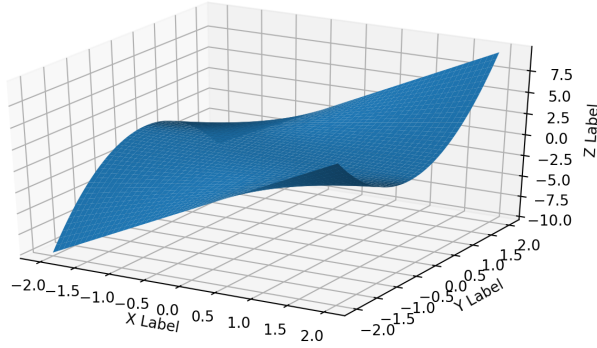
$$\frac{\partial f}{\partial y} = \partial_y f = \lim_{h \rightarrow 0} \frac{f(x, y+h) - f(x, y)}{h} \quad (2)$$

Example problem

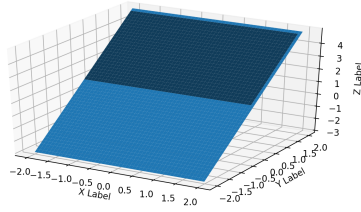
$$f(x, y) = xy^2 + x$$

$$\partial_x f(x, y) = y^2 + 1$$

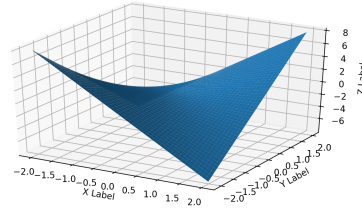
$$\partial_y f(x, y) = 2xy$$



(a)  $f(x, y) = xy^2 + x$



(b)  $\partial_x f(x, y) = y^2 + 1$



(c)  $\partial_y f(x, y) = 2xy$

Figure 1: Example functions

Then there are directional derivatives which are similar to the partial derivatives, but instead of being in a component direction like along the x or y direction it would be in the direction of the vector  $\vec{u}$  with components  $\langle a, b \rangle$ . These derivative can be done in any number of dimensions (4) but for reaction diffusion formulas shown 2D cases (3) matter more.

$$D_u f = \lim_{h \rightarrow 0} \frac{f(x + ha, y + hb) - f(x, y)}{h} \quad (3)$$

$$D_u f(\vec{x}) = \lim_{h \rightarrow 0} \frac{f(\vec{x} + hu) - f(\vec{x})}{h} \quad (4)$$

The directional derivative can also be written as a dot products

$$\begin{aligned} D_u f(x, y) &= a\partial_x f + b\partial_y f \\ &= \langle \partial_x f, \partial_y f \rangle * \langle a, b \rangle \\ &= \langle \partial_x f, \partial_y f \rangle * u \\ &= \nabla f(x, y) * u \end{aligned}$$

This  $\nabla$ , or nabla, is the sign for the grad operator read “del f” This represents a vector of all of the partial derivatives that are possible in a function called the gradient. For example in  $g(x, y, z)$ , grad  $g$ = the partial derivative of  $g$  in respect to  $x$ ,  $y$ , and  $z$ . This gradient should the direction of the fastest increase of the function. So if we had a map of a mountain range the gradient of the map from a point would be the fastest way to get to the top of the nearest mountain but not necessarily the highest mountain. The gradient is often used for optimization problems since it can go “the highest” or most optimal

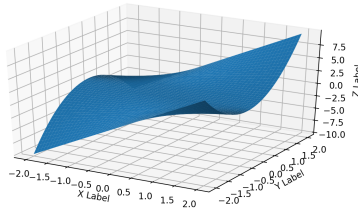
local point in a function of any number of variables. A common abuse of notation is to set  $\nabla$  to a value (5)

$$\nabla = \left\langle \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right\rangle \quad (5)$$

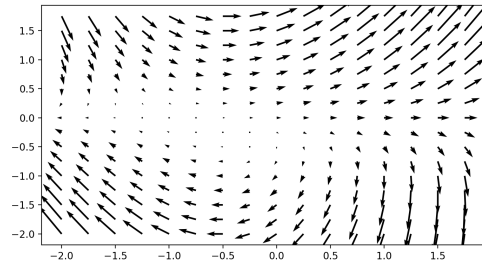
So the notation for the gradient makes sense because you would be treating the function  $f$  as a scalar and multiplying it to the  $\nabla$  vector. This abuse of notation since it would also make sense for the divergence of a vector field. For a vector field the function  $F$  (6) would be a vector function where you could dot product of  $F$  and  $\nabla$  (7)

$$\mathbf{F} = P\hat{i} + Q\hat{j} + R\hat{k} \quad (6)$$

$$\text{div}\mathbf{F} = \nabla \cdot \mathbf{F} = \frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z} \quad (7)$$



(a)  $f(x, y) = xy^2 + x$



(b) the gradient vector field:  $\nabla * f(x, y)$



One can picture a vector field as being a model of fluid flow, where the vector at a point would be the velocity of the fluid. Given this analogy, the divergence if the fluid is incompressible then the amount of fluid that emanates from a point. In a vector field that would be represented by all of the arrows pointing away from a point and the arrows get larger around a point then the divergence  $> 0$  and if the arrows are smaller or point inwards then the divergence  $< 0$ . Both of these tools are useful but when applied together become what is needed for the diffusion part of reaction diffusion that being the laplacian. The laplacian is the divergence of the gradient of a function.

$$\text{Laplacian} = \nabla * (\nabla f) = \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} = \sum_{i=1}^n \frac{\partial^2 f}{\partial x_i^2}$$

$$\nabla^2 = \nabla * \nabla$$

This is called the Laplace operator or the Laplacian of a field because of its relation to Laplace's Equation which has similarities to the heat transfer equation for diffusion. The reason that when put in PDEs they exhibit this diffusion or spreading properly is due to their relation to the second derivative in normal calculus. They identify local maxima and minima but in normal calculus where the second

derivative is 0 in both cases the laplacian is highly positive in minima and highly negative in maxima. In the case of PDEs, that would mean that the values would “move away” from maxima and “fill in” minima which exactly describes osmosis and other diffusion systems.

$$\nabla^2 \mathbf{F} = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} = 0 \quad (8)$$

Equations that satisfies Laplace’s Equation (8) are called harmonic functions where at all points the value of the laplacian = 0

### 3 Diffusion

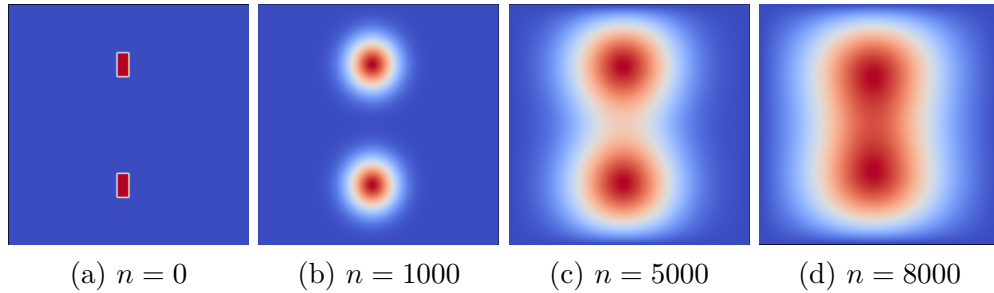
Diffusion is the spread of things from high to low densities. A simple example would be heat or osmosis in cells. In general this is seen as spreading out, like there is diffuse light. The mathematical way this is described is through a partial differential equation using the Laplace operator,  $\nabla^2$ .

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u \quad (9)$$

$$\frac{\partial u}{\partial t} = \alpha \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) \quad (10)$$

This equation 9 is a rather simple partial differential equation that has  $u(x, y, z)$  to determine the heat and  $\alpha$  as the coefficient of thermal diffusivity. Because of the Laplace operator, the equation does not need to show its inputs which also allows for a N-dimensional continuation. In 3 dimensions this equation can be expanded into equation 10.

Figure 3: Heat Diffusion Progression



In this case, (Fig. ??) is progression from 2 point heat sources that spread out and then eventually merge into a single blob that then continues to spread making the entire plane warmer.

The heat diffusion equation (9) uses the laplacian which “pushes” the local maxima down and the local minima up. Later in the reaction

diffusion equations, the laplacian is used to give the equations properties based off their location and physical proximity to other elements. This diffusion component gives reaction diffusion equations their interesting properties since previously converging reactionary systems remain in this possibly unstable equilibrium. The reason for this is that the reactions, spreading due to diffusion, would encounter more fuel as they go until they eventually run out then generating some homeostasis.

## 4 Reaction

In contrast to diffusion equations there are significant differences between different reaction systems. They are called “reaction systems” since they typically contain multiple ordinary or partial differential equations. Were diffusion equations usually have only one variable, reaction systems are often multi variable. There are so many different examples of reaction system, but the SIR and Lotka-Volterra Model.

$$\begin{aligned}
\frac{dS}{dt} &= -\beta s(t)i(t) \\
\frac{dI}{dt} &= -\beta s(t)i(t) - \kappa r(t) \\
\frac{dR}{dt} &= \kappa r(t)
\end{aligned} \tag{11}$$

The SIR model (11), or Susceptible-Infected-Recovered Model, is an computational epidemiology and serves as the core of other more advanced models. In this system,  $\kappa$  represents the days until recovery, and  $\beta$  represents the number of days to be in contact required to spread the disease. This model is intuitive. The recovered is the number of infected times the rate of recovery. The susceptible population is the number of people times the number of infect and the rate of spread. Finally, the infected are the ones that are left over.

## 5 Reaction Diffusion

### References

- P.Gray, S. (1984). Autocatalytic reactions in the isothermal, continuous stirred tank reactor: Oscillations and instabilities in the system  $a + 2b \rightarrow 3b$ ;  $b \rightarrow c$ . *Chemical Engineering Science*, 39(6):1087–1097.
- Turing, A. (1952). The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London*, 237(641):37–72.

## 6 Appendix

Listing 1: Surface

```
1 import numpy as np
2 from mpl_toolkits.mplot3d import Axes3D
3 import matplotlib.pyplot as plt
4
5
6 def fun(x, y):
7     return x*(y**2) + y
8
9
10 def dy(x, y):
11     return 2*x*y
12
13
14 def dx(x, y):
15     return 2*y+1
16
17
18 fig = plt.figure()
19 ax = fig.add_subplot(111, projection='3d')
20 x = y = np.arange(-2.0, 2.0, 0.01)
21 X, Y = np.meshgrid(x, y)
22 zs = np.array([dx(x, y) for x, y in zip(np.ravel(X), np.ravel(Y))])
23 Z = zs.reshape(X.shape)
24
```

```

25 ax.plot_surface(X, Y, Z)
26
27 ax.set_xlabel('X Label')
28 ax.set_ylabel('Y Label')
29 ax.set_zlabel('Z Label')
30
31 plt.show()

```

Listing 2: Gradient

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4
5  def fun(x, y):
6      return x*(y**2) + y
7
8
9  def dy(x, y):
10     return 2*x*y
11
12
13 def dx(x, y):
14     return 2*y+1
15
16
17 x = y = np.arange(-2.0, 2.0, 0.25)
18 X, Y = np.meshgrid(x, y)
19 Dx = np.array([dx(x, y) for x, y in zip(np.ravel(X), np.ravel(Y))])

```



```

20 Dy = np.array([dy(x, y) for x, y in zip(np.ravel(X), np.ravel(Y))])
21 Dx = Dx.reshape(X.shape)
22 Dy = Dy.reshape(Y.shape)
23 Ax = np.array([x + dx for x, dx in zip(np.ravel(X), np.ravel(Dx))])
24 Ay = np.array([y + dy for y, dy in zip(np.ravel(Y), np.ravel(Dy))])
25 Ax = Ax.reshape(X.shape)
26 Ay = Ay.reshape(Y.shape)
27
28 plt.quiver(X, Y, Ax, Ay, cmap=plt.cm.coolwarm)
29
30 plt.show()

```

Listing 3: Diffusion

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Simulation constants
5 alpha = 0.00001 # Laplacian multiplier
6 n = 8000 # number of iterations
7 s = 100 # size of the matrix
8 dx = 2/s # Simulation speed/accuracy
9
10 # Matrix initialization
11 M = np.empty((s, s))
12 M.fill(0)
13 x, y = M.shape
14 # Hotspot Seeding
15 M[20:30, 45:50] = 90

```

```

16 M[70:80, 45:50] = 90
17 np.set_printoptions(precision=1)
18
19
20 # Implementation of the discretized laplacian operator
21 def laplacian(Z):
22     Ztop = Z[0:-2, 1:-1]
23     Zleft = Z[1:-1, 0:-2]
24     Zbottom = Z[2:, 1:-1]
25     Zright = Z[1:-1, 2:]
26     Zcenter = Z[1:-1, 1:-1]
27     return (Ztop + Zleft + Zbottom + Zright - (4 * Zcenter)) / dx**2
28
29
30 # Iterations through PDE
31 for _ in range(n):
32     Mn = M[1:-1, 1:-1]
33     M[1:-1, 1:-1] = Mn + np.multiply(laplacian(M), alpha)
34
35 # Graph Data
36 plt.imshow(M, cmap=plt.cm.coolwarm,
37             interpolation='bilinear', extent=[-1, 1, -1, 1])
38 plt.show()

```

Listing 4: Lotka-Volterra

```

1 import matplotlib.pyplot as plt
2
3 # Model params

```

```

4 alpha = 2/3
5 beta = 4/3
6 gamma = 1
7 delta = 1
8
9 # Simulation constants: discrete time and space
10 dt = 0.001 # the change in time
11 n = 15000 # number of iterations
12
13 # Seed the space
14 U = 1 # Population of Prey
15 V = 0.1 # Population of Predators
16 # output Arrays
17 Uy = []
18 Vy = []
19 X = []
20
21 # Simulate this ODE with Euler's method
22 for i in range(n):
23     # Gets copies to prevent overwriting
24     Uc = U
25     Vc = V
26     # Generate next time step
27     U += (alpha*Uc - beta*Uc*Vc)*dt
28     V += (gamma*Uc*Vc - delta*Vc)*dt
29     # Record outputs
30     Uy.append(U)
31     Vy.append(V)

```

```

32     X.append(i*dt)
33
34 # Graph Data
35 plt.title("Lotka-Volterra Model")
36 plt.plot(X, Uy, 'b', label="pray")
37 plt.plot(X, Vy, 'r', label="preditor")
38 plt.legend()
39 plt.show()

```

Listing 5: SIR

```

1 import matplotlib.pyplot as plt
2
3 # Model params
4 b = 1 # days of contact to spread
5 k = 1/6 # days to recover
6
7 # Simulation params
8 dt = 0.1 # change in time
9 n = 800 # number of iterations
10
11 # Seeding
12 s = 1 # suseptible population
13 i = 1.27E-4 # Infected
14 r = 0 # Recovered
15 # To graph and record values
16 Sy = []
17 Iy = []
18 Ry = []

```

```

19 X = []
20
21 # Simulate ODE with Euler's method
22 for x in range(n):
23     # Edge case
24     if i <= 0:
25         print("Done")
26         break
27     # copy to prevent overwriting
28     Sc = s
29     Ic = i
30     Rc = r
31     # Generate next timestep
32     s += (-b * Sc * Ic) * dt
33     i += ((b * Sc * Ic) - (k * Ic)) * dt
34     r += (k * Ic) * dt
35     # Record values
36     Sy.append(s)
37     Iy.append(i)
38     Ry.append(r)
39     X.append(x*dt)
40
41 # Graph Data
42 plt.title("SIR model")
43 plt.plot(X, Sy, 'g', label="Suseptible")
44 plt.plot(X, Iy, 'r', label="Infected")
45 plt.plot(X, Ry, 'b', label="Recovered")
46 plt.legend()

```

```
47 plt.show()
```