Advection, Diffusion & Reaction Modeling

Marc Los Huertos

February 17, 2024 (ver. 0.4)

Abstract

The movement of compounds in the environment is driven by two processes, advection and diffusion. Of course, these processes occur in three dimensions, but for this class we'll begin with one dimensional processes before getting to more complecated examples.

1 Introduction

Advection and diffusion are two fundamental processes that govern the transport of solutes in the environment. Advection is the process of transport of a solute by the bulk motion of the fluid. Diffusion is the process of transport of a solute by random molecular motion.

1.1 Advection and Convection: Material and Heat

Advection is the transport of a substance by bulk motion. Convection is the transfer of heat by the actual movement of the warmed matter. The equation that are used to describe advection and convection are similar, but the physical processes are different.

1.2 Session Goals

After this sesson, I hope you can do the following:

1. Describe the physical processes of advection and dispersion and reactions
2. Describe the equations used to model A-D-R.
3. Analyze 1-dimensional movement using advection equations
4. Describe Diffusions mathematically
5. Analyze 1-dimensional movement using Fick's Law.
6. Two dimensional analysis of advection

1.3 The Processes and the Equations to Describe Them

1.3.1 Advection

Advection is the process of transport of a solute by the bulk motion of the fluid. The rate of advection is proportional to the velocity of the fluid and the concentration of the solute. The rate of advection is given by the equation:

$$\frac{\partial C}{\partial t} + \nabla \cdot (uC) = 0 \tag{1}$$

where C is the concentration of the solute, t is time, and u is the velocity of the fluid.

1.3.2 Diffusion

Diffusion is the process of transport of a solute by random molecular motion. The rate of diffusion is proportional to the concentration gradient of the solute. The rate of diffusion is given by the equation:

$$\frac{\partial C}{\partial t} = D\nabla^2 C \tag{2}$$

where D is the diffusion coefficient.

1.3.3 Advection-Diffusion Equation

The advection-diffusion equation is a combination of the advection and diffusion equations. The advection-diffusion equation is given by the equation:

$$\frac{\partial C}{\partial t} + \nabla \cdot (uC) = D\nabla^2 C \tag{3}$$

where C is the concentration of the solute, t is time, u is the velocity of the fluid, and D is the diffusion coefficient.

1.3.4 Advection-Difffusion-Reaction Equation

The advection-diffusion-reaction equation is a combination the advection, diffusion, and reaction equations. The advection-diffusion-reaction equation is given by the equation:

$$\frac{\partial C}{\partial t} + \nabla \cdot (uC) = D\nabla^2 C + R \tag{4}$$

where C is the concentration of the solute, t is time, u is the velocity of the fluid, D is the diffusion coefficient, and R is the reaction term.

1.3.5 Advenction-Difffusion-Reaction in multi-phase systems and for shapes with variable geometry

The advection-diffusion-reaction equation can be extended to multi-phase systems and to shapes with variable geometry. The advection-diffusion-reaction equation for multi-phase systems and for shapes with variable geometry is given by the equation:

$$\frac{\partial C}{\partial t} + \nabla \cdot (uC) = D\nabla^2 C + R \tag{5}$$

where C is the concentration of the solute, t is time, u is the velocity of the fluid, D is the diffusion coefficient, and R is the reaction term.

2 Applications using R

2.1 R as a Calculator and Modeling Environment

We can use R to solve the advection-diffusion equation. The following code uses the 'deSolve' package to solve the advection-diffusion equation for a simple one-dimensional case.

```
# Load the deSolve package
library(deSolve)

# Define the advection-diffusion equation
advection_diffusion <- function(t, C, parms) {
  with(as.list(parms), {
    dC <- D * (diff(C, lag = 2) - 2 * diff(C, lag = 1) + diff(C, lag = 0)) / dx^2
    dC[1] <- 0
    dC[n] <- 0
    list(dC)
  })
}

# Set the parameters
parms <- list(
  D = 0.1,  # Diffusion coefficient
  dx = 0.1  # Spatial step
)

# Set the initial conditions
C0 <- c(0, rep(0, 98), 1, rep(0, 98), 0)

# Set the times at which to evaluate the solution
times <- seq(0, 100, by = 1)
```

```
# Solve the advection-diffusion equation

out <- ode(y = C0, times = times, func = advection_diffusion, parms = parms)

# Plot the solution

plot(out, xlab = "Distance", ylab = "Concentration", type = "l")
```

2.2 1D Transportion Model

The 'ReacTran' package provides a function to solve the advection-diffusion-reaction equation for a simple one-dimensional case.

```
# Load the ReacTran package
library(ReacTran)

## Loading required package:  rootSolve
## Loading required package:  deSolve
## Loading required package:  shape
```

```
tran.1D(C = 1, D = 0, flux.up = 1, v = 5, A= 1, dx = 1, full.output = TRUE)

## $dC
## [1] -4
##
## $C.up
## [1] 0.2
##
## $C.down
## [1] 1
##
## $dif.flux
## [1] 0 0
##
## $adv.flux
## [1] 1 5
##
## $flux
## [1] 1 5
##
## $flux.up
## [1] 1
##
## $flux.down
## [1] 5
```

2.3 Solving a 1-D reaction tranport mdodel

```
library(ReacTran)
out <- steady.1D(func = advModel, y = runif(25), params = parms, nspace= 1, positive = TRUE)
```

We can use R to solve the advection-diffusion-reaction equation. The following code uses the 'deSolve' package to solve the advection-diffusion-reaction equation for a simple one-dimensional case.

```
# Load the deSolve package
library(deSolve)

# Define the advection-diffusion-reaction equation
advection_diffusion_reaction <- function(t, C, parms) {
  with(as.list(parms), {
    dC <- D * (diff(C, lag = 2) - 2 * diff(C, lag = 1) + diff(C, lag = 0)) / dx^2 - k * C
    dC[1] <- 0
    dC[n] <- 0
    list(dC)
  })
}

# Set the parameters
parms <- list(
  D = 0.1,  # Diffusion coefficient
  dx = 0.1,  # Spatial step
  k = 0.01  # Reaction rate
)

# Set the initial conditions
C0 <- c(0, rep(0, 98), 1, rep(0, 98), 0)

# Set the times at which to evaluate the solution
times <- seq(0, 100, by = 1)

# Solve the advection-diffusion-reaction equation
out <- ode(y = C0, times = times, func = advection_diffusion_reaction, parms = parms)

# Plot the solution
plot(out, xlab = "Distance", ylab = "Concentration", type = "l")
```

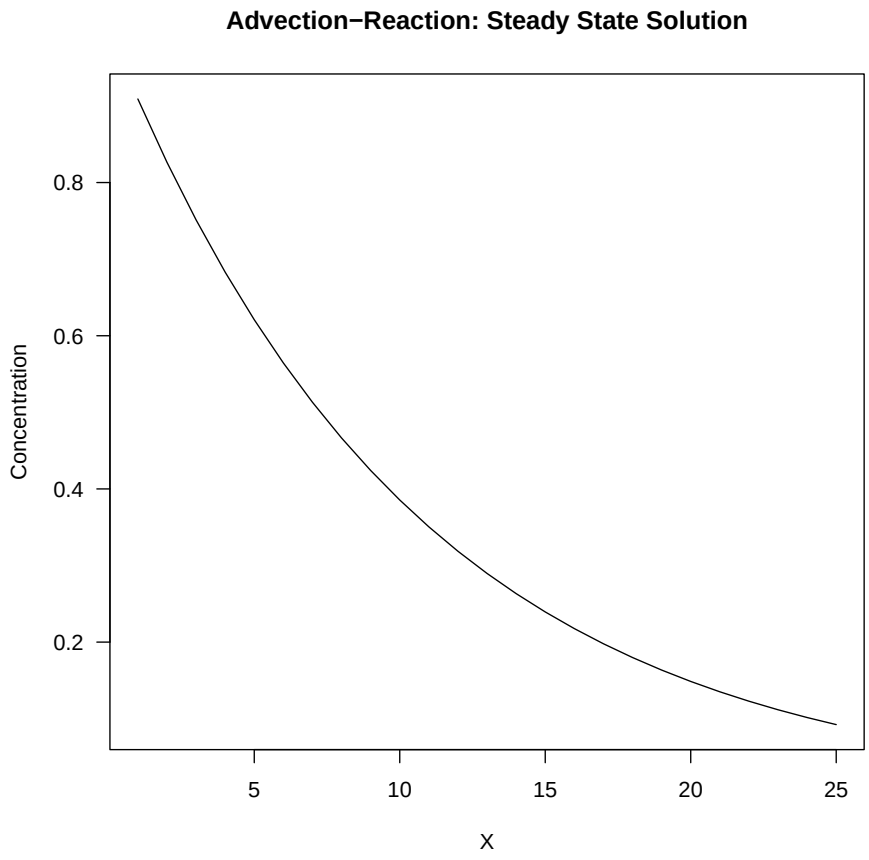
2.4 1-D Reaction-Transport Model

```
library(ReacTran)

parms <- c(F0 = 1, v=1, k = 0.1, D = 0, dx = 1)

advModel <- function(t, C, parms) {
  with(as.list(parms), {
    Tran <- tran.1D(C = C, D = D, flux.up = F0, v = v, dx = dx)
    Consumption = k * C
    dC <- Tran$dC - Consumption

    return(list(dC = dC, Consumption = Consumption, flux.up = Tran$flux.up, flux.down = Tran
  })
}

out <- steady.1D(func = advModel, y = runif(25), parms = parms, nspec= 1, positive = TRUE)

parms <- c(F0 = 1, v=1, k = 0.5, D=0, dx = 1)
out2 <- steady.1D(func = advModel, y = runif(25), parms = parms, nspec= 1, positive = TRUE)

parms <- c(F0 = 1, v=1, k = 0.5, D=50, dx = 1)
out3 <- steady.1D(func = advModel, y = runif(25), parms = parms, nspec= 1, positive = TRUE)
```

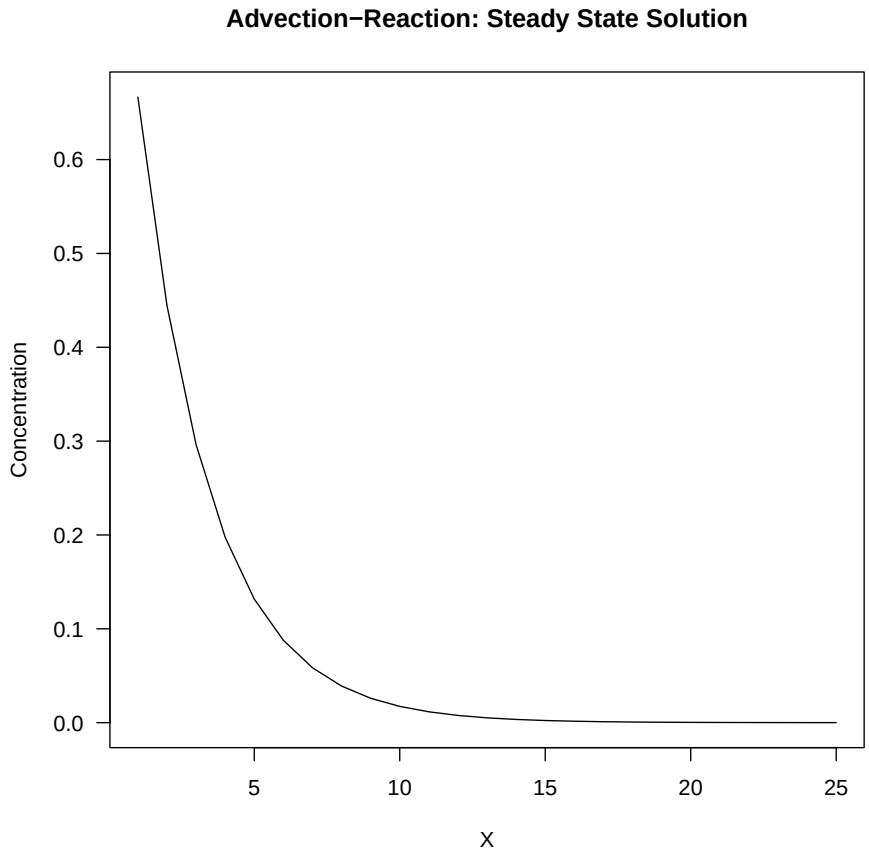
```
out

## $y
##  [1] 0.90909091 0.82644628 0.75131480 0.68301346 0.62092132 0.56447393
##  [7] 0.51315812 0.46650738 0.42409762 0.38554329 0.35049390 0.31863082
## [13] 0.28966438 0.26333125 0.23939205 0.21762914 0.19784468 0.17985880
## [19] 0.16350800 0.14864363 0.13513058 0.12284598 0.11167816 0.10152561
## [25] 0.09229601
##
## $Consumption
##  [1] 0.090909091 0.082644628 0.075131480 0.068301346 0.062092132 0.056447393
##  [7] 0.051315812 0.046650738 0.042409762 0.038554329 0.035049390 0.031863082
## [13] 0.028966438 0.026333125 0.023939205 0.021762914 0.019784468 0.017985880
## [19] 0.016350800 0.014864363 0.013513058 0.012284598 0.011167816 0.010152561
## [25] 0.009229601
##
## $flux.up
## [1] 1
##
## $flux.down
## [1] 0.09229601
##
## attr(,"precis")
```

```
## [1] 3.806766e-01 2.201006e-09
## attr(,"steady")
## [1] TRUE
## attr(,"class")
## [1] "steady1D"  "rootSolve" "list"
## attr(,"dimens")
## [1] 25
## attr(,"nspec")
## [1] 1
```

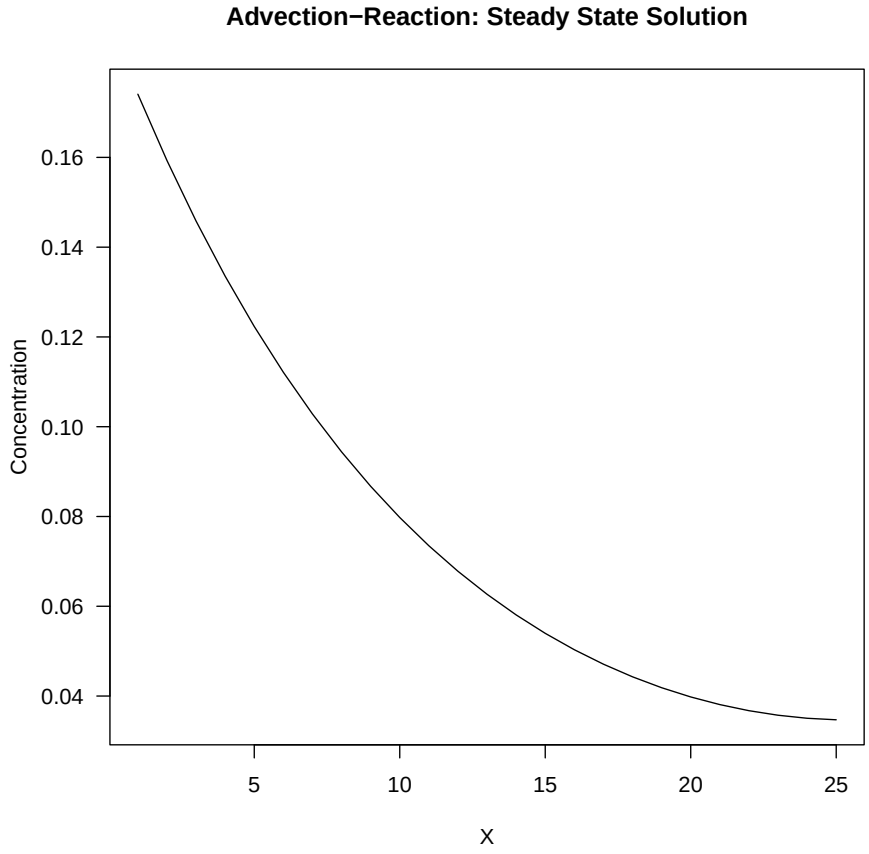
```
par(mfrow=c(1,3))
plot(out, xlab = "X", ylab = "Concentration", las=1, main="Advection-Reaction: Steady State
```



```
plot(out2, xlab = "X", ylab = "Concentration", las=1, main="Advection-Reaction: Steady State
```




```
plot(out3, xlab = "X", ylab = "Concentration", las=1, main="Advection-Reaction: Steady State
```



2.5 Oxygen Consumption Porous Spherical Particle

We will be modeling the consumption of oxygen in a "sand-sized" porous spherical particle. The model is based on the following equation:

$$\frac{\partial C}{\partial t} = -v\frac{\partial C}{\partial x} - k(C)$$

where C is the concentration of oxygen, D is the diffusion coefficient, v is the velocity of the fluid, and $k(C)$ is the rate of oxygen consumption.

At this scale the velocity will be zero. Thus, we will rely on diffusion to for the oxygen movement to where it is consumed.

We start with defining the size and porosity of the particle and use R to create a grid to solve the advection-diffusion-reaction equation.

We will create a grid to model the particle with Radius R and N $(= 100)$ grid points. We will also define the properties of the particle such as porosity,

Table 1: Chararacteristics of the Particle

Parameter	Description	Typical Range	Modelled Value
R	Radius of the particle	$0.005 - 0.2$ cm	0.025 cm
Porosity	Proportion of void space	$0.005 - -0.7$	0.7

diffusion coefficient (D = 400), and the rate of oxygen consumption, $R_{02} = 10^6$.

Although we are modeling a one-dimensional system, we will need to create a grid surface as a circle to effectively model the particle surface area changes as O2 diffuses into the particle and is consumed by the reactions in the particle.

```
grid <- setup.grid.1D(x.up=0, L = R, N = N)

por.grid <- setup.prop.1D(value=por, grid=grid)
D.grid <- setup.prop.1D(value=D, grid=grid)

sphere.surf <- function(x) 4*pi*x^2
A.grid <- setup.prop.1D(func=sphere.surf,  grid=grid)
```

Finally, we need to define the O2 concentration at the surface of the particle and the O2 consumption rate of the particle.

Table 2: Oxygen Consumption in the Particle

Parameter	Description	Typical Range	Modelled Value
C_{ow}	Concentration of O2 in Water	$0.1 - 0.3$ μmols/cm^{-3}	0.25 μmols/cm^{-3}
R_{02}	Rate of oxygen consumption	$10^5 - 10^6$ μmols/cm^{-3} /year	10^6 μmols/cm^{-3}/year
K_s	O2 saturation	$0.001 - 0.01$ μmols/cm^{-3}	0.005 μmols/cm^{-3}

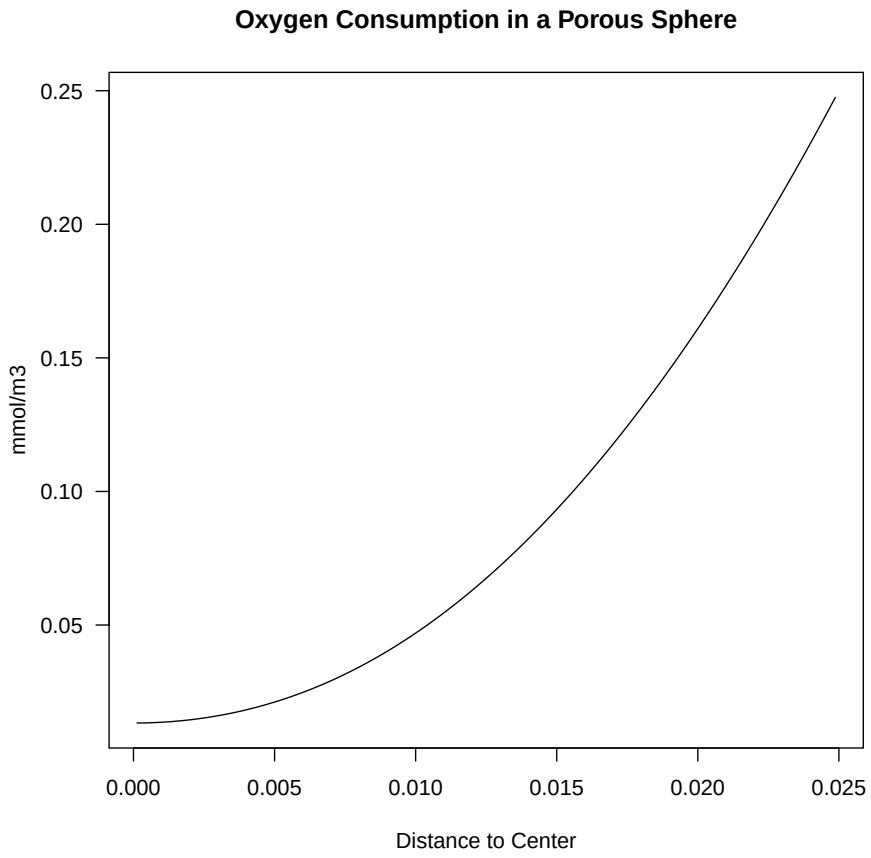
Note, we often measure oxygen using ppm (parts per million), but the model uses μmols/cm^{-3}. The conversion is 1 ppm = 0.0224 μmols/cm^{-3}, thus, we are modeling Ks within a range of 2.24 - 6.72 ppm, using Ks = 0.22 μmols/cm^{-3}.

Next we create a function to model the oxygen consumption in the, particle that relies on We will use the `tran.1D` function to solve the advection-diffusion equation and the `steady.1D` function to solve the steady state solution of the advection-diffusion-reaction equation.

```
Aggregate.Model <- function(time, O2, pars) {
  tran <- tran.1D(C = O2, C.down = C.ow.02, D = D.grid,
          A=A.grid, VF = por.grid, dx = grid)
    reac <- - R.02 * (O2 /(Ks + O2))
    return(list(dCdt= tran$dC + reac, reac = reac,
                flux.up=tran$flux.up, flux.down=tran$flux.down))
}
```

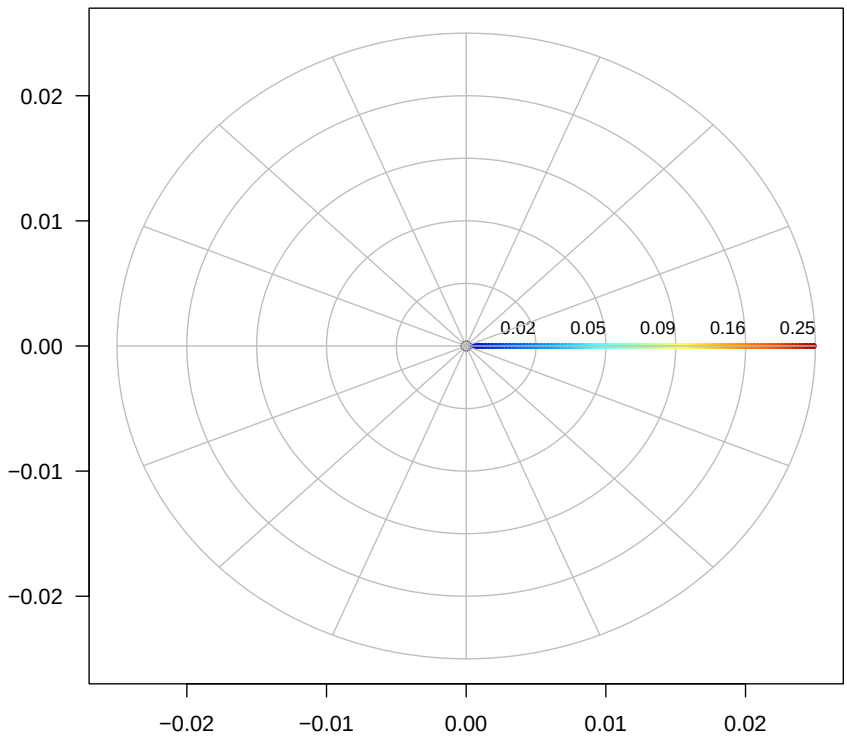
```
O2.agg <- steady.1D(y = runif(N), func=Aggregate.Model,
                    nspec=1, positive=TRUE, atol = 1e-10)
```

Figure 1: Oxygen Consumption in a Porous Sphere



The plot shows the oxygen concentration in the particle. The concentration is highest at the surface and decreases as it moves into the particle. The concentration is zero at the center of the particle.

```
diffusion2D <- function(t,conc,par){
Conc <- matrix(nr=n,nc=n,data=conc) # vector to 2-D matrix
dConc <- -r*Conc*Conc # consumption
BND <- rep(1,n) # boundary concentration
```




```
# constant production in certain cells
dConc[ii]<- dConc[ii]+ p

#diffusion in X-direction; boundaries=imposed concentration

Flux <- -Dx * rbind(rep(0,n),(Conc[2:n,]-Conc[1:(n-1),]),rep(0,n))/dx
dConc <- dConc - (Flux[2:(n+1),]-Flux[1:n,])/dx

#diffusion in Y-direction
Flux <- -Dy * cbind(rep(0,n),(Conc[,2:n]-Conc[,1:(n-1)]),rep(0,n))/dy
dConc <- dConc - (Flux[,2:(n+1)]-Flux[,1:n])/dy

return(list(as.vector(dConc)))
```

```
}
```

After specifying the values of the parameters, 10 cells on the 2-D grid where there will be substance produced are randomly selected (ii).

14 Package rootSolve : roots, gradients and steady-states in R 0.0 0.2 0.4 0.6 0.8 1.0 0.0 0.2 0.4 0.6 0.8 1.0 2-D diffusion+production x y Figure 5: Steady-state solution of the nonlinear 2-Dimensional model

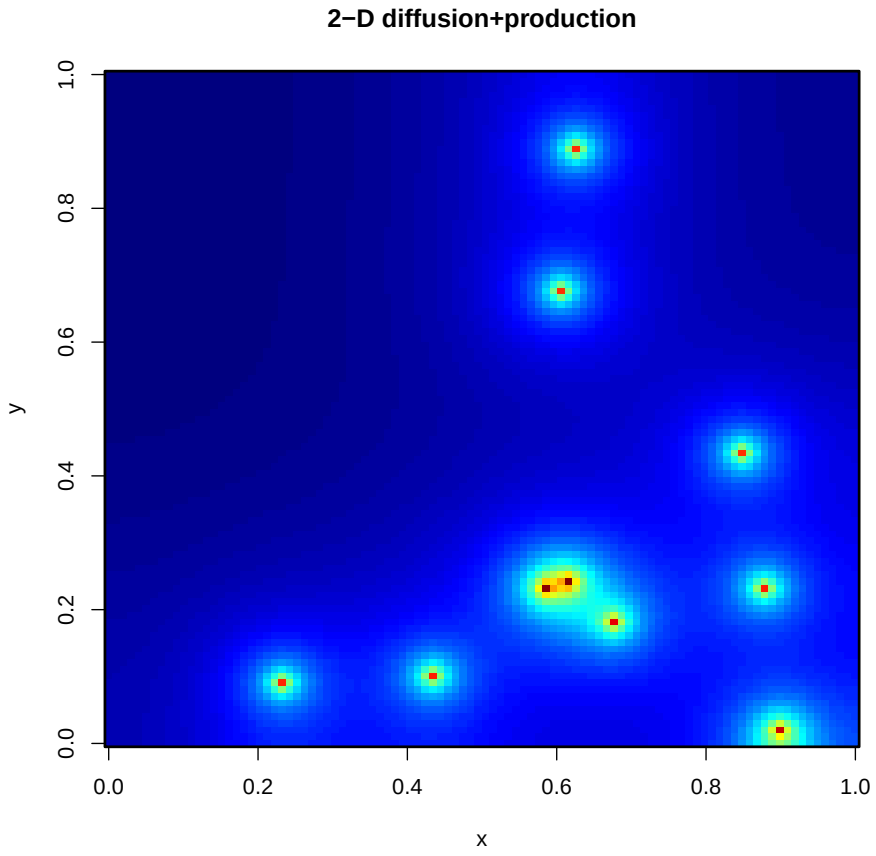
```
# parameters
dy <- dx <- 1 # grid size
Dy <- Dx <- 1.5 # diffusion coeff, X- and Y-direction
r <- 0.01 # 2-nd-order consumption rate (/time)
p <- 20 # 0-th order production rate (CONC/t)
n <- 100
# 10 random cells where substance is produced at rate p
ii <- trunc(cbind(runif(10)*n+1,runif(10)*n+1))
```

The steady-state is found using function steady.2D. It takes as arguments a.o. the dimensionality of the problem (dimens) and lrw=1000000, the length of the work array needed by the solver. If this value is set too small, the solver will return with the size needed. It takes about 0.5 second to solve this 10000 state variable model.

```
Conc0 <- matrix(nr=n,nc=n,10.)
print(system.time(
ST3 <- steady.2D(Conc0,func=diffusion2D,parms=NULL,pos=TRUE,dimens=c(n,n),
lrw=1000000,atol=1e-10,rtol=1e-10,ctol=1e-10)
))

##    user  system elapsed
##   0.261   0.026   0.287
```

user system elapsed 1.044 0.032 1.076 The S3 image method is used to generate the steady-state plot.

```
image(ST3,main="2-D diffusion+production", xlab="x", ylab="y")
```



3 Considering 2D Models

2.4. Steady-state solution of 2-D PDEs Function steady.2D effciently snds the steady-state of 2-dimensional problems. Karline Soetaert 13 In the following model @C @t = Dx @2C @x2 + Dy @2C @y2 .. r C2 + pxy a substance C is consumed at a quadratic rate (r C2), while dispersing in X- and Y-direction. At certain positions (x,y) the substance is produced (rate p). The model is solved on a square (100*100) grid. There are zero- ux boundary conditions at the 4 boundaries. The term Dx @2C @x2 is in fact shorthand for: .. @Flux @x where Flux = ..Dx @C @x i.e. it is the negative of the ux gradient, where the ux is due to diffusion. In the numerical approximation fo the ux, the concentration gradient is approximated as the subtraction of two matrices, with the columns or rows shifted (e.g. Conc[2:n,]-Conc[1:(n-1),]). The ux gradient is then also approximated by subtracting entire matrices (e.g. Flux[2:(n+1),]-Flux[1:(n),]).

This is very fast. The zero- ux at the boundaries is imposed by binding a column or row with 0-s.

4 Conclusion