Examples of usage

Pablo Rodríguez-Sánchez

2019-03-05

Installation

To install the package, please type

```
devtools::install_github("PabRod/consRvative", ref = "develop")
```

in your *R* console.

One dimensional examples

Allee effect

A single-species population dynamics model with Allee effect is governed by the following differential equation:

$$\frac{dN}{dt} = rN\left(\frac{N}{A} - 1\right)\left(1 - \frac{N}{K}\right)$$

It is easy to see that this differential equation has three equilibrium points, $N = 0$, $N = K$ and $N = A$, being all of them stable but the latter one, which is unstable. We'll use the parameters $r = 1$, $A = 0.5$ and $K = 1$.

```
r <- 1
A <- 0.5
K <- 1

f <- function(x) { r * x * (x/A - 1) * (1 - x/K) }
```

We can use our method `approxPot1D` to approximate the potential function at a set of points. First, we have to create the points.

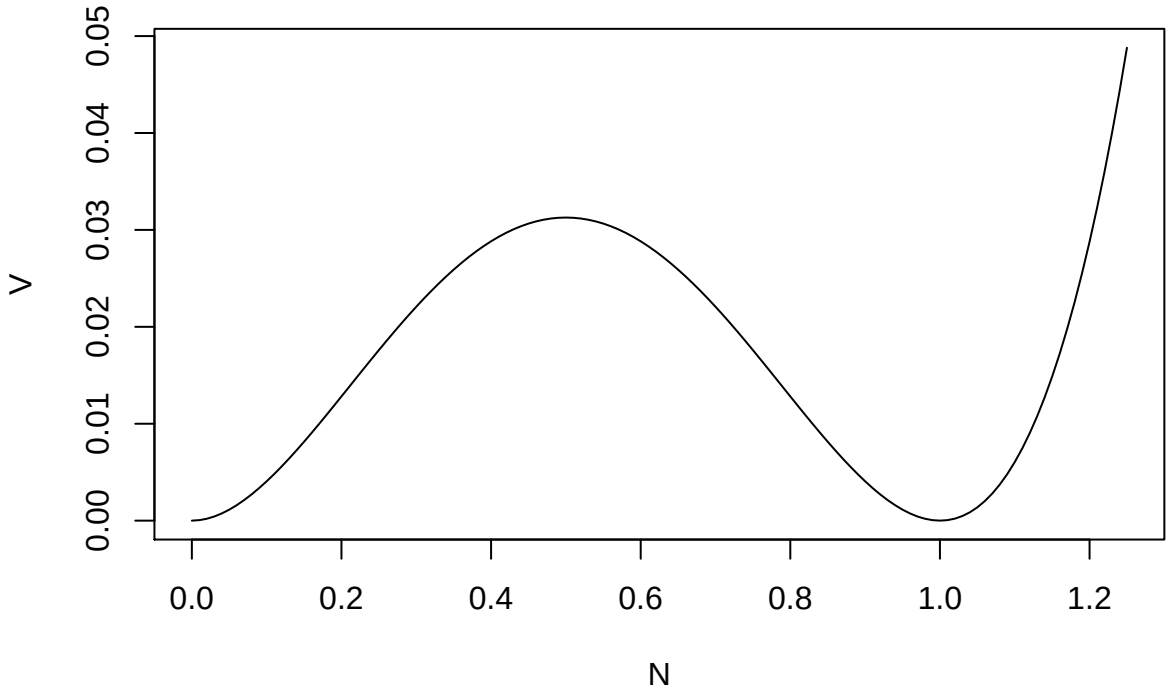
```
xs <- seq(0, 1.25, by = 0.01)
```

and then pass them to our algorithm:

```
Vs <- approxPot1D(f, xs)
```

By plotting the result, we clearly see that the two stable equilibria appear at $N = 0$ and $N = K = 1$, and the unstable one at $N = A = 0.5$, as we expected.

```
plot(xs, Vs,
     type = 'l', xlab = 'N', ylab = 'V')
```



Two dimensional examples

In this section we'll apply our method to a collection two dimensional systems.

We'll use the following auxiliary function to comfortably generate the plots:

```
plotResults <- function(xs, ys, result, aspect = 1)
{
  # Store the pseudo-potential as X, Y and Z columns in the data dataframe
  dataV <- expand.grid(X = xs, Y = ys)
  dataV$Z <- as.vector(result$V)

  # And generate the potential plot...
  plotV <- levelplot(Z ~ X*Y, data = dataV, scales = list(draw = TRUE),
                     col.regions = colorRamps::matlab.like,
                     aspect = aspect, xlab = 'x', ylab = 'y',
                     main = 'Approximate potential')

  # Do the same for the error
  dataErr <- expand.grid(X = xs, Y = ys)
  dataErr$Z <- as.vector(result$err)
```

```
  plotErr <- levelplot(Z ~ X*Y, data = dataErr, scales = list(draw = TRUE),
                       col.regions = colorRamps::green2red,
                       aspect = aspect, xlab = 'x', ylab = 'y',
                       at = seq(0, 1, by = 0.05),
                       main = 'Relative error')

  # Plot both together
  print(plotV, split = c(1, 1, 2, 1), more = TRUE)
  print(plotErr, split = c(2, 1, 2, 1), more = FALSE)
}
```

Synthetic examples

We generated some abstract, synthetic examples in order to test our method. Here we present some of them.

A gradient system: the four well potential

In this section we'll deal with the two-dimensional differential equation given by:

$$\begin{cases} \frac{dx}{dt} = f(x,y) = -x(x^2-1) \\ \frac{dy}{dt} = g(x,y) = -y(y^2-1) \end{cases}$$

This is a gradient system (because $\frac{\partial f}{\partial x} = \frac{\partial g}{\partial y}$ everywhere). This means that the gradient - curl decomposition will have zero curl term everywhere and, thus, there exists a well defined potential. Particularly, the potential can be analytically proven to be:

$$V(x,y) = \frac{x^2}{4}(x^2-2) + \frac{y^2}{4}(y^2-2) + V_0$$

Let's try to compute it using our algorithm. First, we'll code our function as a vector:

```
f <- function(x) {c(-x[1]*(x[1]^2 - 1),
                    -x[2]*(x[2]^2 - 1))}
```

Our region of interest is now two-dimensional. We need, thus, two vectors to create our grid of points:

```
xs <- seq(-1.5, 1.5, by = 0.025)
ys <- seq(-1.5, 1.5, by = 0.025)
```

Now we are ready to apply `approxPot2D`:

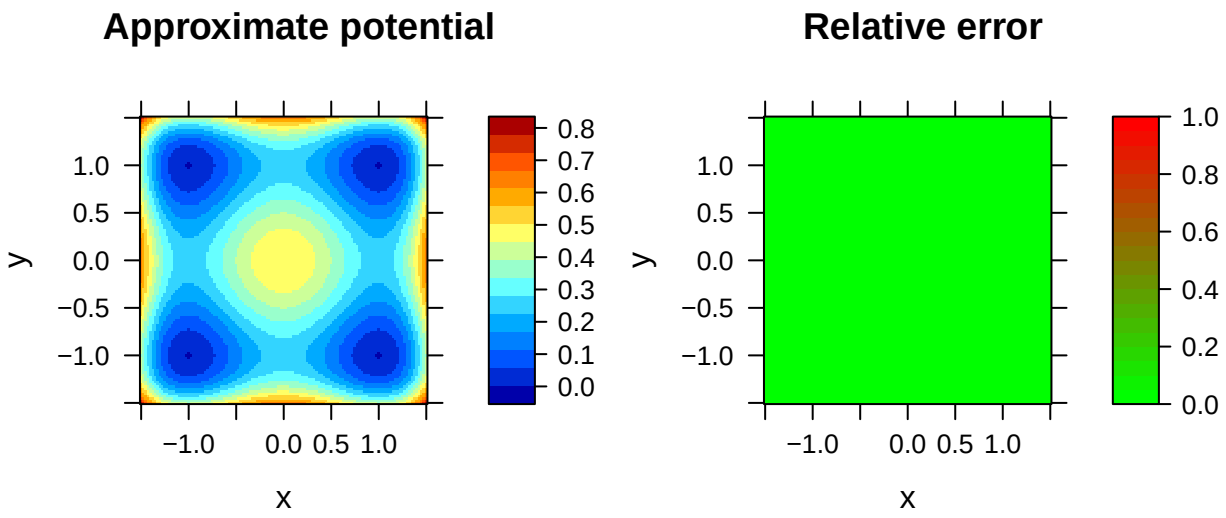
```
result <- approxPot2D(f, xs, ys)
```

`result` is a list that contains two fields:

- `result$V` contains the estimated values of the potentials at each grid point
- `result$err` contains the estimated error at each grid point

By plotting them we see:

```
plotResults(xs, ys, result)
```



Provided our example is a purely gradient system, it should not surprise us that the error is zero everywhere.

```
max(result$err) == 0
```

```
## [1] TRUE
```

A non-gradient system

In this example we will apply our algorithm to the system given below:

$$\begin{cases} \frac{dx}{dt} = f(x,y) = -y \\ \frac{dy}{dt} = g(x,y) = x \end{cases}$$

This is an extreme case. The gradient - curl decomposition will give us zero gradient part everywhere. Let's feed our algorithm with these example to see what happens:

First, we code the dynamics in vector form:

```
# Dynamics
f <- function(x) {c(-x[2],
                    x[1])}
```

Secondly, we define our region of interest:

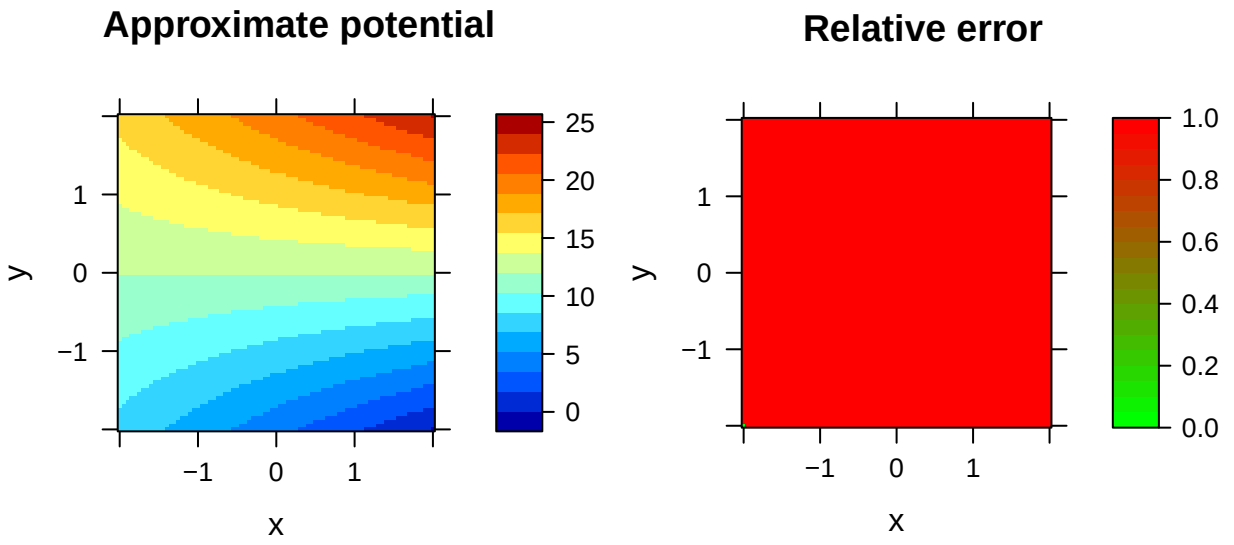
```
xs <- seq(-2, 2, by = 0.05)
ys <- seq(-2, 2, by = 0.05)
```

And then we are ready to apply our algorithm:

```
result <- approxPot2D(f, xs, ys)
```

The resulting approximate potential is plotted below. Being this a purely non-gradient system we expect our pseudopotential to not be trustworthy. By calculating the error we can see that actually that's the case.

```
plotResults(xs, ys, result)
```



The fact that the underlying equations are non-gradient have been captured by the algorithm.

Biological examples

Here we apply our methods to some dynamical equations well known in biology. While the abstract equations in the previous sections can be manipulated to increase or decrease their curl to gradient ratio, equations describing natural dynamical systems don't allow such a manipulation. Once again, the error map will let us know if our system allows a pseudopotential or not.

Simple regulatory gene network

A bistable network cell fate model can be described by the set of equations:

$$\begin{cases} \frac{dx}{dt} = f(x,y) = b_x - r_x x + \frac{a_x}{k_x + y^n} \\ \frac{dy}{dt} = g(x,y) = b_y - r_y y + \frac{a_y}{k_y + x^n} \end{cases}$$

Such a system represents two genes (x and y) that inhibit each other. This circuit works as a toggle switch with two stable steady states, one with dominant x , the other with dominant y.

We can code it in vector form:

```
# Parameters
bx <- 0.2
ax <- 0.125
kx <- 0.0625
rx <- 1

by <- 0.05
ay <- 0.1094
ky <- 0.0625
ry <- 1

n <- 4

# Dynamics
f <- function(x) {c(bx - rx*x[1] + ax/(kx + x[2]^n),
                    by - ry*x[2] + ay/(ky + x[1]^n))}
```

This set of equations is, in general, not gradient (because $\frac{\partial f}{\partial y} \neq \frac{\partial g}{\partial x}$). Anyways, we can use the method `approxPot2D` to compute the approximate potential.

First, we need to define our region of interest:

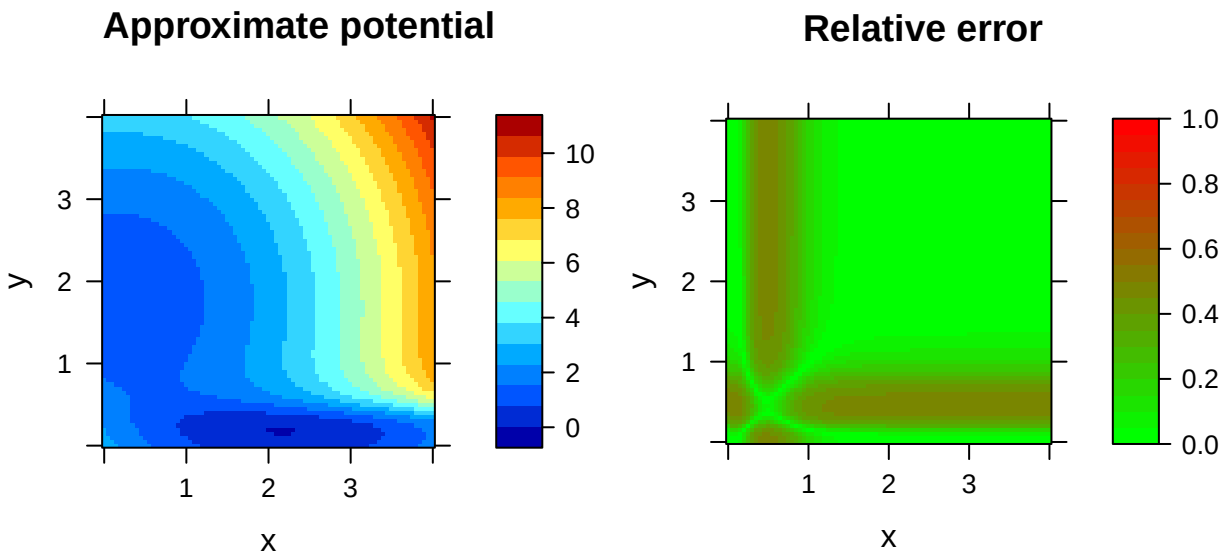
```
xs <- seq(0, 4, by = 0.05)
ys <- seq(0, 4, by = 0.05)
```

And then we are ready to apply our algorithm:

```
result <- approxPot2D(f, xs, ys)
```

The resulting approximate potential is plotted below. Being this not a gradient system it is advisable to plot also the estimated error. The areas in green represent small approximation error, so the potential can be safely used in those regions.

```
plotResults(xs, ys, result)
```



Lotka-Volterra predator prey dynamics

Here we will use a variation of the classical Lotka-Volterra predator prey model. Particularly, the Rosenzweig-MacArthur model, that adds a function g accounting for predator saturation and a carrying capacity k to the prey growth.

The dynamics, being x the prey biomass and y the predator biomass, look like:

$$\begin{cases} \frac{dx}{dt} = f(x,y) = rx(1-\frac{x}{k}) - g(x)xy \\ \frac{dy}{dt} = g(x,y) = eg(x)xy - my \end{cases}$$

with $g(x)$ being a saturation function:

$$g(x) = \frac{1}{h+x}$$

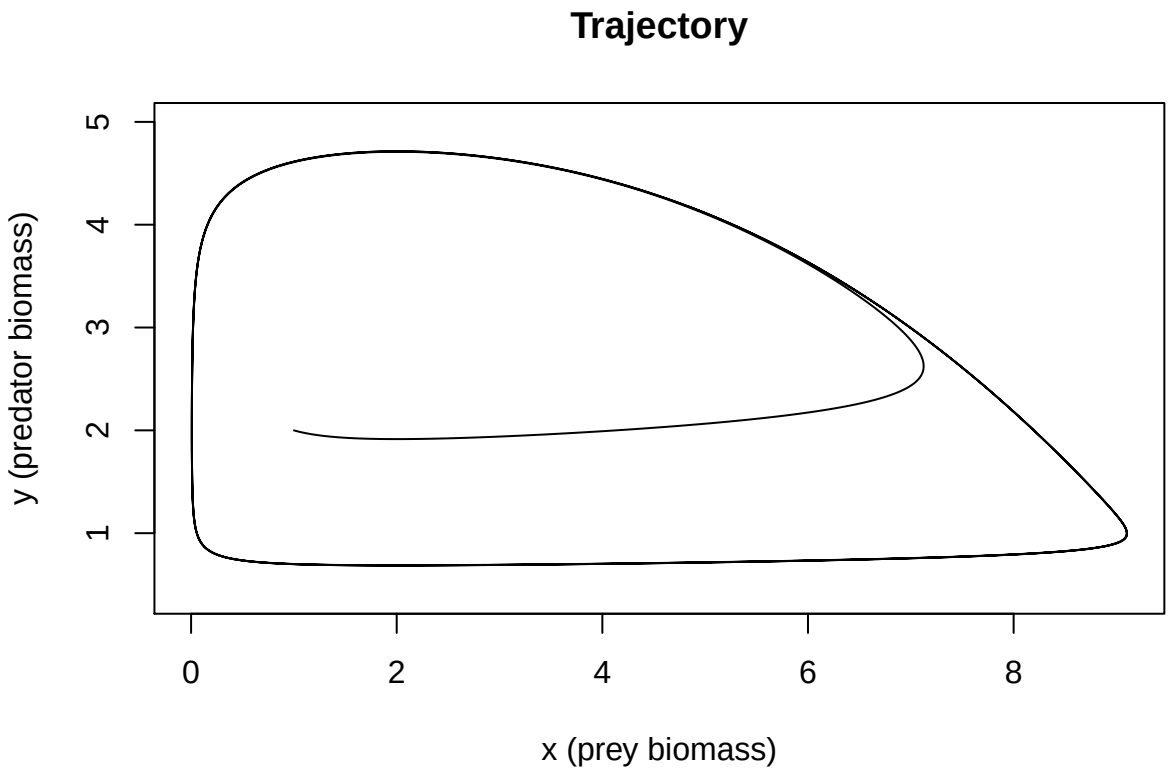
We can code it in vector form:

```
# Parameters
r <- 1
k <- 10
h <- 2
e <- 0.2
m <- 0.1
```

```r
# Auxiliary function
g <- function(x) {1/(h + x)}

# Dynamics
f <- function(x) {c(r*x[1]*(1 - x[1]/k) -g(x[1])*x[1]*x[2],
                    e*g(x[1])*x[1]*x[2] - m*x[2])}
```

Such a system has a limit cycle attractor, as we can see simulating one of its trajectories (particularly, the one beginning at $x = 1$ and $y = 2$):



For such a system, we expect our pseudopotential to have a high error over large portions of the phase plane. Let's check it. First, we need to define our region of interest:

```r
xs <- seq(0, 10, by = 0.05)
ys <- seq(0, 5, by = 0.05)
```

And then we are ready to apply our algorithm:

```r
result <- approxPot2D(f, xs, ys)
```

Even for highly non-gradient systems, our method will compute some pseudopotential. By plotting the estimated error, we can see that it is high almost everywhere. This means that our algorithm noticed that the system is highly non-gradient, and thus, the previously computed pseudopotential is of very limited use.

```
plotResults(xs, ys, result, aspect = 0.5)
```