Interactions from function

Stefano Allesina

2022-07-09

Setting

We have a pool of n species, with dynamics governed by GLV:

$$\frac{dx_i}{dt} = x_i r_i (1 - \sum_j B_{ij} x_j)$$

We perform experiments in which a subset k of species are grown together. If the species can stably coexist (either at equilibrium, or in a limit cycle or chaotic attractor), their time-averaged abundance will be:

$$x^{(k)} = \left(B^{(k)}\right)^{-1} 1$$

where $x^{(k)}$ is a vector containing the densities of the species in k, and $B^{(k)}$ is the sub-matrix of B in which we have retained only the rows/cols corresponding to the species in k.

We perform several experiments, each time varying the initial composition. For each experiment, we measure a function $f^{(k)}$ (for example, total biomass, respiration, production of a certain metabolite). We make a strong assumption:

$$f^{(k)} = \sum_{j \in k} \alpha_j x_j^{(k)}$$

The goal of the project is to predict $f^{(k)}$ for novel communities, when we have learned the relevant parameters from a set of experiments.

Change of variables

We have:

$$f^{(k)} = \sum_{i,j} \alpha_i (B^{(k)})^{-1}_{ij} = (\alpha^{(k)})^T (B^{(k)})^{-1} 1$$

Call $BD(\frac{1}{\alpha}) = Q$. Then

$$f^{(k)} = 1^T (B^{(k)} D(\frac{1}{\alpha^{(k)}}))^{-1} 1 = 1^T D(\alpha^{(k)}) (B^{(k)})^{-1} 1 = 1^T (Q^{(k)})^{-1} 1$$

Because we always have that $f^{(k)}$ is a quadratic form, only the symmetric part of $(Q^{(k)})^{-1}$ will matter. As such, the simplest case to study is that in which Q is symmetric.

Learning the parameters for symmetric Q

Suppose $n = 3$, and that we have performed experiments spanning all the $2^n - 1$ combinations. We want to learn

$$Q = \begin{pmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{pmatrix}$$

Analyzing the three experiments with a single species we get:

$$(Q^{(1)})^{-1} = \frac{1}{q_{11}} = f^{(1)}$$
$$(Q^{(2)})^{-1} = \frac{1}{q_{22}} = f^{(2)}$$
$$(Q^{(3)})^{-1} = \frac{1}{q_{33}} = f^{(3)}$$

Solving:

$$q_{11} = \frac{1}{f^{(1)}}$$
$$q_{22} = \frac{1}{f^{(2)}}$$
$$q_{33} = \frac{1}{f^{(3)}}$$

Now take the two-species experiments:

$$f^{(i,j)} = 1^T \begin{pmatrix} q_{ii} & q_{ij} \\ q_{ij} & q_{jj} \end{pmatrix}^{-1} 1 = 1^T \begin{pmatrix} \frac{1}{f^{(i)}} & q_{ij} \\ q_{ij} & \frac{1}{f^{(j)}} \end{pmatrix}^{-1} 1 = \frac{-2f^{(i)}f^{(j)}q_{ij} + f^{(i)} + f^{(j)}}{1 - f^{(i)}f^{(j)}q_{ij}^2}$$

We can solve for q_{ij}, finding two solutions:

$$q_{ij} = \frac{1 \pm \sqrt{\frac{(f^{(i,j)} - f^{(i)})(f^{(i,j)} - f^{(j)})}{f^{(i)}f^{(j)}}}}{f^{(i,j)}}$$

Given that we have three pairs i, j, we end up with 8 possible solutions. Problem: how do we choose the right solution? If we have a solution for all the coefficients in Q, we can attempt predicting the function $f^{(1,2,3)}$.

Example code

```
set.seed(12)
n <- 3
# make Q symmetric
B <- matrix(-rnorm(n * n), n, n)
alpha <- runif(n)
Q <- B %*% diag(1 / alpha)
Q <- Q + t(Q)
B <- Q %*% diag(alpha)
```

```
# compute function for each sub-community, using either B, alpha or Q
results <- matrix(0, 0, n + 2)
for (i in 1:(2^n - 1)){
  presence <- as.numeric(intToBits(i)[1:n])
  Bk <- B[presence > 0, presence > 0, drop = FALSE]
  alphak <- alpha[presence > 0]
  Qk <- Q[presence > 0, presence > 0, drop = FALSE]
  results <- rbind(results, c(presence, alphak %*% rowSums(solve(Bk)), sum(solve(Qk))))
}
# note that the results are the same
print(results)
```

```
##      [,1] [,2] [,3]       [,4]       [,5]
## [1,]    1    0    0 0.22480562 0.22480562
## [2,]    0    1    0 0.02820799 0.02820799
## [3,]    1    1    0 0.22808610 0.22808610
## [4,]    0    0    1 1.02554574 1.02554574
## [5,]    1    0    1 0.08559187 0.08559187
## [6,]    0    1    1 3.94500592 3.94500592
## [7,]    1    1    1 0.08865733 0.08865733
```

```
# solve system  of equations
f <- as.complex(results[,n+1]) # the part under sqrt can be negative...
f1 <- f[1]
f2 <- f[2]
f3 <- f[4] # this is spp 3 by itself
f12 <- f[3] # this is spp 1 and 2
f13 <- f[5]
f23 <- f[6]
k11 <- 1/f1
k22 <- 1/f2
k33 <- 1/f3
k12 <- (1 + sqrt((f12 - f1) * (f12 - f2) / (f1 * f2))) / f12 # choose +
k13 <- (1 - sqrt((f13 - f1) * (f13 - f3) / (f1 * f3))) / f13 # choose -
k23 <- (1 + sqrt((f23 - f2) * (f23 - f3) / (f2 * f3))) / f23 # choose +
K <- matrix(c(k11, k12, k13,
              k12, k22, k23,
              k13, k23, k33), 3, 3, byrow = TRUE)
# we have recovered all the coefficients
print(Q)
```

```
##          [,1]      [,2]      [,3]
## [1,] 4.448287  5.794128 2.8813657
## [2,] 5.794128 35.450948 5.2931958
## [3,] 2.881366  5.293196 0.9750906
```

```
print(Re(K))
```

```
##          [,1]      [,2]      [,3]
## [1,] 4.448287  5.794128 2.8813657
## [2,] 5.794128 35.450948 5.2931958
## [3,] 2.881366  5.293196 0.9750906
```

```
# we can therefore predict the function for all spp together
sum(solve(Re(K)))
```

```
## [1] 0.08865733
```

Generic Q

```
get_results <- function(Q){
  n <- nrow(Q)
  results <- matrix(0, 0, n + 1)
  for (i in 1:(2^n - 1)){
    presence <- as.numeric(intToBits(i)[1:n])
    Qk <- Q[presence > 0, presence > 0, drop = FALSE]
    results <- rbind(results, c(presence, sum(solve(Qk))))
  }
  return(results)
}
```

Can we learn Q when it is not symmetric?

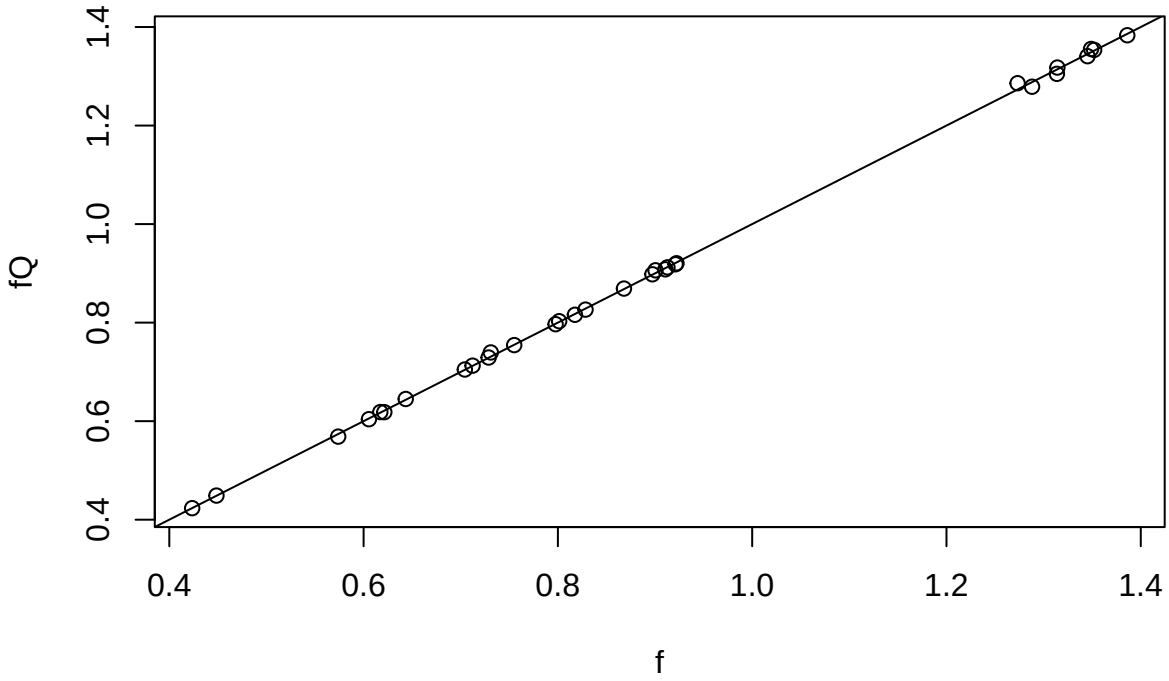
```
set.seed(13)
n <- 5
# make Q non symmetric
Q <- abs(matrix(rnorm(n * n), n, n))
diag(Q) <- diag(Q) + 1 # strong diagonal
results <- get_results(Q)
design_matrix <- results[,1:n]
f <- results[,n+1]

get_ssq <- function(pars, design_matrix, f, returnf = FALSE){
  n <- sqrt(length(pars))
  Q <- matrix(pars, n, n)
  fQ <- f
  for (i in 1:length(f)){
    presence <- design_matrix[i,]
    Qk <- Q[presence > 0, presence > 0, drop = FALSE]
    fQ[i] <- sum(solve(Qk))
  }
  if (returnf) return(fQ)
  return(sum((f - fQ)^2))
}

hc <- function(bestpars, design_matrix, f, nsteps = 500, startpert = 0.25){
  bestres <- get_ssq(bestpars, design_matrix, f)
  dev <- startpert
  for (j in 1:10){
    for (i in 1:nsteps){
      tmppar <- bestpars * (1 + rnorm(length(bestpars)) * dev)
      tmpres <- get_ssq(tmppar, design_matrix, f)
      if (tmpres < bestres){
        bestpars <- tmppar
        bestres <- tmpres
        #print(tmpres)
      }
    }
    dev <- dev / 2
  }
```

```
  return(bestpars)
}

tmp <- list(par = as.vector(diag(rep(1,n)))) # start with identity matrix
for (iter in 1:10){
  print(iter)
  tmp$par <- hc(tmp$par, design_matrix, f)
  tmp$value <- get_ssq(tmp$par, design_matrix, f)

  tmp <- optim(par = tmp$par, fn = get_ssq, design_matrix = design_matrix, f = f,
               method = "Nelder-Mead", control = list(maxit = 5000, trace = FALSE))
  print(tmp$value)
  print(tmp$value)
  tmp <- optim(par = tmp$par, fn = get_ssq, design_matrix = design_matrix, f = f,
               method = "BFGS", control = list(maxit = 5000, trace = FALSE))
  print(tmp$value)
}
```

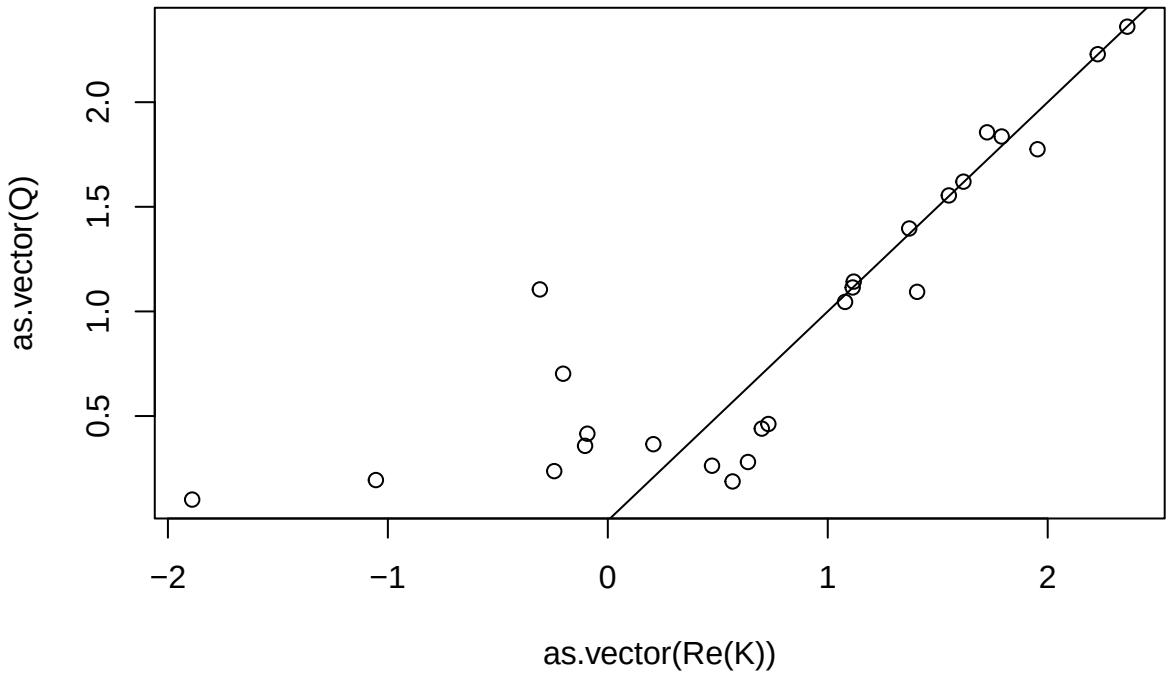
```
## [1] 1
## [1] 0.0407151
## [1] 0.0407151
## [1] 0.0005895434
## [1] 2
## [1] 0.0005895304
## [1] 0.0005895304
## [1] 0.0005895304
## [1] 3
## [1] 0.0005895237
## [1] 0.0005895237
## [1] 0.0005895237
## [1] 4
## [1] 0.0005895179
## [1] 0.0005895179
## [1] 0.0005895179
## [1] 5
## [1] 0.0005895091
## [1] 0.0005895091
## [1] 0.000589509
## [1] 6
## [1] 0.0005894953
## [1] 0.0005894953
## [1] 0.0005894953
## [1] 7
## [1] 0.0005894886
## [1] 0.0005894886
## [1] 0.0005894886
## [1] 8
## [1] 0.0005894867
## [1] 0.0005894867
## [1] 0.0005894867
## [1] 9
## [1] 0.0005894841
## [1] 0.0005894841
## [1] 0.0005894841
```

```
## [1] 10
## [1] 0.0005894827
## [1] 0.0005894827
## [1] 0.0005894827
```

```
fQ <- get_ssq(tmp$par, design_matrix, f, TRUE)
plot(f, fQ)
abline(c(0,1))
```




```
K <- matrix(as.vector(tmp$par), n, n)
plot(as.vector(Re(K)), as.vector(Q))
abline(c(0,1))
```


as.vector(Q)
0.5
1.0
1.5
2.0
−2
−1
0
1
2
as.vector(Re(K))