THE PROGRAMMING OF THE HOMOLOGY CALCULATION OF DIHEDRAL QUANDLES R VERSION

ANSGAR WENZEL

Abstract. This paper aims to serve both as a documentation and an explanation of the implementation in R of the algorithm for the calculation of the homology of racks, quandles and their biversions as detailed in [1]

1. Introduction

This paper is divided into two parts. The first chapter comprises the commented code. The second chapter explains the thinking behind the implementation, in particular where there are differences to the algorithm defined in [1]. The notation does also follow the notation in the aforementioned paper. Please note that this will hopefully also be coded in C in order to a) speed up the runtime and b) lower memory usage.

2. The code

The code is organised into (currently) three different files. The first consists of two subroutines for the calculation of the Smith Normal Form and to find the matrix X. Then there is the main function to calculate the homology. The input, degree, k, degenerate, are used for dihedral quandles in particular and only used for the calculation of the boundary matrices [1] and the subsequent output of the calculation result.

```r
#first load the packages necessary for third-party functions.
library(Matrix)
library(schoolmath)
library(MASS)
library(numbers)

#first define the subfunctions

#this is used to calculate the left elementary matrix X
findX <- function(A){
  X <- diag(nrow(A))    #create Identity matrix
  G_X <- cbind(A,X)     #combine identity matrix with original matrix for Gaussian
      elimination
  GX_res <- GaussianElimination(G_X)  #Gaussian elimination
  X <- GX_res[, (ncol(GX_res)-nrow(A)+1):ncol(GX_res)]  #extract the left matrix, X
```

Date: June 27, 2013.

[1]to do

```
  return(X)
}

#this calculates the smith normal form based on the calculation of the Hermite
    Normal Form.
#In particular, HNF((HNF(A))^T)
smith <- function(A){
  HF <- hermiteNF(A)$H
  S <- hermiteNF(t(HF))$H
  return(S)
}
#here is the main function to calculate the homology
homology <- function(degree,k,degenerate){
  boundary_F <- matrix(nrow=12,ncol=6,byrow=T,data=c
      (1,-1,0,0,1,0,1,-1,-1,0,0,0,-1,1,1,0,0,0,-1,1,0,0,-1,0,0,0,1,-1,0,1,-1,0,1,-1,0,0,0,0,-1,1,0,
      )#a matrix - Matrix(,sparse=TRUE)
  boundary_G <- matrix(c(-1,0,1,-1,1,0,0,-1,1,1,-1,0,0,1,-1,1,0,-1),ncol=3,nrow=6,
      byrow=T)#another matrix
  rho <- rankMatrix(boundary_G)[1]  #first, this calculates the rank of the matrix G
      . This removes the need to calculate D and Y later.
  q <- nrow(boundary_G)
  r <- ncol(boundary_G)
  q_rho <- q - rho
  X <- findX(boundary_G)
  Z <- X[(rho + 1):q, ]        #only take the rows that map to zero (i.e. only the
      boundaries)
  B <- GaussianElimination(boundary_F)  #identify the cycles, i.e. remove the
      boundaries via Gaussian elimination
  B <- B[which(rowSums(abs(B))!=0), ]   #remove the boundaries in order to lower
      memory allocation
  N <- round(Z %*% ginv(B)) #calculate N. Details in documentation
  S <- smith(N)
  Delta <- diag(S)  #Extract the values necessary for the output.
  s <- length(Delta)
  l <- 0
  ones <- 0
  output <- c()
  for(i in 1:s){
    if(Delta[i]==1){
      ones <- ones + 1    #count number of ones
    }
    else if(Delta[i]!=0){
      l <- l + 1
      output <- append(output,Delta[i])    #count and extract nonzero and non-one
          values in the diagonal
    }
    else{
      break
    }
  }

  #the following is the output depending on the number of zeroes.
  if(s>l+ones){#check if there are any values not equal to one or zero
```

```
      print(paste0("The ",degree,ifelse((degree%%10)==1,"st",ifelse((degree%%10)==2,
          "nd",ifelse((degree%%10)==3,"rd","th")))," homology group of R_",k," is
          isomorphic to Z^",s-(k+ones)," plus the following:"))
  } else{
    print(paste0("The ",degree,ifelse((degree%%10)==1,"st",ifelse((degree%%10)==2,"
        nd",ifelse((degree%%10)==3,"rd","th")))," homology group of R_",k," is
        isomorphic to the following:"))
  }
  if(l>0){ #if so, print out the resulting Z_n groups
    for(i in 1:l){
      print(paste0("Z_",Delta[ones+i],ifelse(i!=l," plus","")))
    }
  }
  else{
    print("0")
  }
}
```

. ../R_code/homology_calculation.R

The second file collects two functions written by other people. In particular, the function for Gaussian Elimination is used.

```
#this is a collection of other peoples function, used for the homology calculation.
#in particular, the Gaussian elimination is used.

GaussianElimination <- function(A, B, tol=sqrt(.Machine$double.eps),
                                verbose=FALSE, fractions=FALSE){
  # A: coefficient matrix
  # B: right-hand side vector or matrix
  # tol: tolerance for checking for 0 pivot
  # verbose: if TRUE, print intermediate steps
  # fractions: try to express nonintegers as rational numbers
  # If B is absent returns the reduced row-echelon form of A.
  # If B is present, reduces A to RREF carrying B along.
  if (fractions) {
    mass <- require(MASS)
    if (!mass) stop("fractions=TRUE needs MASS package")
  }
  if ((!is.matrix(A)) || (!is.numeric(A)))
    stop("argument must be a numeric matrix")
  n <- nrow(A)
  m <- ncol(A)
  if (!missing(B)){
    B <- as.matrix(B)
    if (!(nrow(B) == nrow(A)) || !is.numeric(B))
      stop("argument must be numeric and must match the number of row of
           A")
    A <- cbind(A, B)
  }
  i <- j <- 1
  while (i <= n && j <= m){
    while (j <= m){
      currentColumn <- A[,j]
      currentColumn[1:n < i] <- 0
      # find maximum pivot in current column at or below current row
      which <- which.max(abs(currentColumn))
      pivot <- currentColumn[which]
      if (abs(pivot) <= tol) { # check for 0 pivot
        j <- j + 1
        next
      }
      if (which > i) A[c(i, which),] <- A[c(which, i),]  # exchange rows
      A[i,] <- A[i,]/pivot            # pivot
      row <- A[i,]
      A <- A - outer(A[,j], row)      # sweep
      A[i,] <- row                    # restore current row
      if (verbose) if (fractions) print(fractions(A))
      else print(round(A, round(abs(log(tol,10)))))
      j <- j + 1
      break
    }
    i <- i + 1
  }
  # 0 rows to bottom
  zeros <- which(apply(A[,1:m], 1, function(x) max(abs(x)) <= tol))
  if (length(zeros) > 0){
    zeroRows <- A[zeros,]
    A <- A[-zeros,]
    A <- rbind(A, zeroRows)
  }
  rownames(A) <- NULL
  if (fractions) fractions (A) else round(A, round(abs(log(tol, 10))))
}

rref <- function(A, tol=sqrt(.Machine$double.eps),verbose=FALSE,
                 fractions=FALSE){
  ## A: coefficient matrix
  ## tol: tolerance for checking for 0 pivot
  ## verbose: if TRUE, print intermediate steps
  ## fractions: try to express nonintegers as rational numbers
  ## Written by John Fox
  if (fractions) {
    mass <- require(MASS)
    if (!mass) stop("fractions=TRUE needs MASS package")
  }
  if ((!is.matrix(A)) || (!is.numeric(A)))
    stop("argument must be a numeric matrix")
  n <- nrow(A)
  m <- ncol(A)
  for (i in 1:min(c(m, n))){
    col <- A[,i]
    col[1:n < i] <- 0
    # find maximum pivot in current column at or below current row
    which <- which.max(abs(col))
    pivot <- A[which, i]
    if (abs(pivot) <= tol) next     # check for 0 pivot
    if (which > i) A[c(i, which),] <- A[c(which, i),]  # exchange rows
    A[i,] <- A[i,]/pivot            # pivot
    row <- A[i,]
    A <- A - outer(A[,i], row)      # sweep
    A[i,] <- row                    # restore current row
    if (verbose)
      if (fractions) print(fractions(A))
    else print(round(A,round(abs(log(tol,10)))))
  }
  for (i in 1:n)
    if (max(abs(A[i,1:m])) <= tol)
      A[c(i,n),] <- A[c(n,i),] # 0 rows to bottom
  if (fractions) fractions (A)
  else round(A, round(abs(log(tol,10))))
}
```

. ../R_code/found_functions.R

The third file contains the function to calculate boundary matrices for (bi)racks and (bi)quandles based on the Switch equation and in $\mathbb{Z}_k$, in particular the dihedral quandles. The up and down actions are called in their respective functions and can easily be changed. [2]

```
#this collects all the functions necessary for the calculation of boundary matrices
    of (bi)quandles

up_action <- function(a,b,k){
  result <- (2 * b - a) %% k
  return(result)
}

down_action <- function(a,b,k){
  return(a)
}
```

. ../R_code/boundary_calculations.R

3. Explanation

The code mostly follows the algorithm as laid out in [1]. There are a few differences, as follows:

- We do calculate neither XGY nor Y itself. This is due to the fact that XGY is only used as a proxy to calculate the rank and that Y is never used again. We think it faster and more memory effective to calculate the rank directly.
- We do not have XGY with D in the lower right, but rather in the top left corner. Thus, we do not use the first $q - \rho$ rows but rather the last $q - \rho$ rows.
- We also use Gaussian elimination to calculate the span of the rowspace of F or Z, depending on Notation.
- The N ¡- round(Z %*% ginv(B)) equation [3] follows by simple arithmetic.
- Finally, instead of coding a Smith Normal Form Algorithm (of which there are plenty!), we use the Hermitian Normal Form function from the numbers package in R.

[2]this is still in progress

[3]%*% is matrix multiplication in R, ¡- is equivalent to = and ginv is the general inverse of a mxn matrix

4. Used Software

This programme was written in R and used the following packages:

- R base installation, version 3.0.1 (2013-05-16) ”Good Sport”, [2].
- Matrix, [3].
- schoolmath, [4].
- MASS, [5].
- numbers, [6].

References

[1] R. Fenn, *How to Calculate Homology*, 2013, unpublished.

[2] R Core Team (2013). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

[3] D. Bates & M. Maechler (2013). Matrix: Sparse and Dense Matrix Classes and Methods. R package version 1.0-12. http://CRAN.R-project.org/package=Matrix.

[4] J. Schlarmann & J. Wienand (2012). schoolmath: Functions and datasets for math used in school. R package version 0.4. http://CRAN.R-project.org/package=schoolmath

[5] W. N. Venables & B. D. Ripley (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

[6] H. W. Borchers (2012). numbers: Number-theoretic Functions. R package version 0.3-3. http://CRAN.R-project.org/package=numbers