ex05

May 25, 2023

1 Introduction to computation physics ex. 5

1.0.1 Paris J. Huth: Gruppe 1

1.0.2 Q inich Pakal Figueroa Coc: Gruppe 5

1. Gaussian elimination

[63]:
```
# implementing check for tridiagonal elements
is.tridia <- function(A) {
    n <- nrow(A)

    for (i in 1:(n-1)) {
        if (A[i+1, i] == 0 || A[i, i+1] == 0){
            return(FALSE)}
    }

    return(TRUE)
}

# function for gaussian elemination
tridiagaus <- function(A, b) {
    n <- nrow(A)

    if (!is.tridia(A))
        stop("Matrix A is not tridiagonal.")
    # Check if the dimensions of matrix A and vector b are valid
    if (ncol(A) != n || length(b) != n) {
        stop("Invalid input dimensions.")
    }

    # Forward elimination
    for (i in 2:n) {
        m <- A[i, i-1] / A[i-1, i-1]
        A[i, i] <- A[i, i] - m * A[i-1, i]
        b[i] <- b[i] - m * b[i-1]
    }

    return(list(A = A, b = b))
```

```
}
```

[64]:

```
# test
A <- matrix(c(2, -1, 0, -1, 2, -1, 0, -1, 2), nrow = 3, ncol = 3)
b <- c(1, 2, 3)

solution <- tridiagaus(A, b)
print(solution)
```

```
$A
     [,1] [,2]      [,3]
[1,]    2 -1.0  0.000000
[2,]   -1  1.5 -1.000000
[3,]    0 -1.0  1.333333

$b
[1] 1.000000 2.500000 4.666667
```

2. Backward substitution

[65]:

```
# b
# implement check for upper triangular structure
is.uppertri <- function(A) {
    n <- nrow(A)

    for (i in 2:n) {
        if (any(A[i, 1:(i-1)] != 0))
            return(FALSE)
    }

    return(TRUE)
}

# Backwards substitution
backsub <- function(A, b) {
    n <- length(b)

    # Check if A is an upper triangular matrix
    if (!is.uppertri(A))
        stop("Matrix A is not upper triangular.")

    x <- numeric(n)
    x[n] <- b[n] / A[n, n]

    for (i in (n-1):1) {
        x[i] <- (b[i] - sum(A[i, (i+1):n] * x[(i+1):n])) / A[i, i]
    }
```

```
    return(x)
}
```

[66]:
```
# test
A <- matrix(c(2, 0, 0, -1, 2, 0, 0, -1, 2), nrow = 3, ncol = 3)
b <- c(1, 2, 3)

solution <- backsub(A, b)
print(solution)
```

```
[1] 1.375 1.750 1.500
```

3. given the vectors a, b, c and y calculate x

[67]:
```
# c
# since input is required to be passed as vectors, application of previous␣
↪functions
# need costruction of a matrix, but handling with vectors is easier in r
tridiagSolver <- function(a, b, c, y) {
    n <- length(y)
    x <- numeric(n)

    # Check if the lengths of input vectors are valid
    if (length(a) != (n-1) || length(b) != n || length(c) != (n-1)) {
        stop("Invalid input vectors.")
    }

    # Forward elimination
    for (i in 2:n) {
        m <- a[i-1] / b[i-1]
        b[i] <- b[i] - m * c[i-1]
        y[i] <- y[i] - m * y[i-1]
    }

    # Backward substitution
    x[n] <- y[n] / b[n]

    for (i in (n-1):1) {
        x[i] <- (y[i] - c[i] * x[i+1]) / b[i]
    }

    return(x)
}
```

4. Example case

[68]:
```
# d

N <- 10
a <- rep(-1, N-1)
b <- rep(3/2, N)
c <- rep(-1, N-1)
M <- matrix(c(rep(c(3/2,-1,rep(0,8),-1),9),3/2),ncol=10,byrow=TRUE)

y <- rep(1/10, N)

solution <- tridiagSolver(a, b, c, y)
print(solution)
```

```
 [1]  0.09565217  0.04347826 -0.13043478 -0.33913043 -0.47826087 -0.47826087
 [7] -0.33913043 -0.13043478  0.04347826  0.09565217
```

5. Checking deviation

[69]:
```
# e
# using initial function tridiagaus requires a matrix
M <- matrix(c(rep(c(3/2,-1,rep(0,8),-1),9),3/2),ncol=10,byrow=TRUE)

solution <- tridiagSolver(a, b, c, y)# savgin the result
d <- tridiagaus(M,solution)# inputing in original matrix solver
ds <- abs(d$b-y)

x <- cbind(y,d$b,ds)
colnames(x) <-c('y', 'b', 'd')
x
mean(ds)
```

y	b	d
0.1	0.09565217	0.004347826
0.1	0.10724638	0.007246377
0.1	-0.00173913	0.101739130
0.1	-0.34492754	0.444927536
0.1	-0.29011858	0.390118577
0.1	-0.62009662	0.720096618
0.1	-0.95241280	1.052412805
0.1	-1.99429640	2.094296400
0.1	4.40746803	4.307468031
0.1	1.29065945	1.190659455

1.03133127554236

Reinserting the resulting vector from tridiagSolver leads to an average deviation of 1.03 from the intput vector.