Matriz de Hilbert

En álgebra lineal, una matriz de Hilbert, introducida por Hilbert(1894), es una matriz cuadrada cuyas entradas son:En álgebra lineal, una matriz de Hilbert, introducida por Hilbert(1894), es una matriz cuadrada cuyas entradas son:

$$H_{1j} = \frac{1}{i+j-1}, \quad i,j = 1,\ldots,n$$

Además, su inversa se puede expresar de forma cerrada como:

$$H_{i,j}^{-1} = (-1)^{i+j}(i+j-1)\binom{n+i-1}{n-j}\binom{n+j-1}{n-i}\binom{i+j-2}{i-1}^2, \quad i,j = 1,\ldots,n$$

Problema 1

Para esta práctica deberán hacer un código que devuelva la matriz de Hilbert para cualquier n, por ejemplo para n = 5.

Solución

Primero observamos que la matriz de Hilbert será una matriz cuadrada dado que sus índices i,j van de 1 hasta n. Creamos una matriz cuadrada de nxn y luego usamos dos loops para iterar elemento por elemento realizando la operacion $H_{i,j}$

Listing 1: *Imprimimos nuestra matriz con* $n = 5$

```
matriz_hilbert <- function(n){
  matriz = matrix(nrow = n, ncol = n)
  for(i in 1:n){
    for(j in 1:n){
      matriz[i,j] = 1/(i+j-1)
    }
  }
  return(matriz)
}
print(matriz_hilbert(5))
```

Salida:

```
             [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 1.0000000 0.5000000 0.3333333 0.2500000 0.2000000
[2,] 0.5000000 0.3333333 0.2500000 0.2000000 0.1666667
[3,] 0.3333333 0.2500000 0.2000000 0.1666667 0.1428571
[4,] 0.2500000 0.2000000 0.1666667 0.1428571 0.1250000
[5,] 0.2000000 0.1666667 0.1428571 0.1250000 0.1111111
```

Problema 2

Y encontrar la n tal que, no se puede hayar numéricamente la inversa de H. Finalmente explicar por qué no se puede.

Solución

```
for(i in 1:100){
  if (det(matriz_hilbert(i)) == 0) {
    print(i)  # Imprimime 29
    break
  }
}
```

Sabemos que para que una matriz tenga inversa su determinante debe ser diferente de cero, por lo tanto, haciendo un loop de 1 hasta un numero muy grande, nos fijamos que si al sacar el determinante de la matriz de hilbert nos devuelve un cero, significa que hemos encontrada esa n, en este caso, al hacer la prueba encontramos que cuando $n = 29$ se rompe el ciclo implicando que no tiene matriz inversa.

Regla de Simpson

Hay situaciones en las que es imposible encontrar el valor exacto de una integral definida, en tal caso ser a necesario usar algún método numérico, para haya una aproximación.

La regla de Simpson es un método que a partir de polinomios de grado dos, aproxima el area abajo la curva.

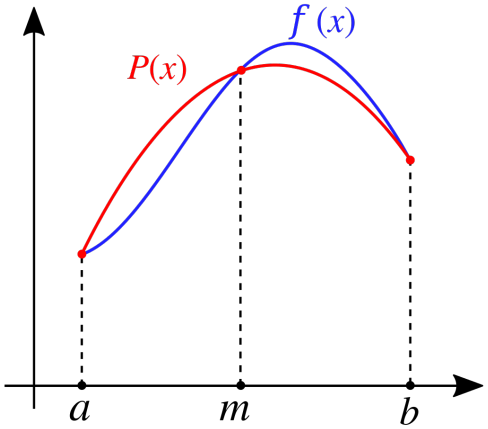

Fig. 1 – *Regla de Simpson: https://es.wikipedia.org/wiki/Regla_de_Simpson*

Si el intervalo $[a,b]$ se divide en n subintervalos, y n es un número par, la regla viene dada por la siguiente expresión:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 2\sum_{i=2,4,5}^{n-2} f(x_i) + 4\sum_{i=1,3,5}^{n-1} f(x_i) + f(x_n)\right]$$

Con $h = \frac{b-a}{n}$ y $x_i = a + ih$.

Si $|f^{(4)}(x)| \leq K$ para $a \leq x \leq b$. Si E_s es el error relacionado con la regla de Simpson, entonces:

$$|E_s| \leq \frac{K(b-a)^5}{n^4}$$

Problema 3

Para este ejercicio deberán escribir una función que tenga como input: f, a, b, n y como output: el valor aproximado de $\int_a^b f(x)dx$ usando la regla de Simpson.

Probar su código con funciones simples como $f(x) = x^2$,$f(x) = \sin(x)$ y no tan simples como $f(x) = e^{-x^2}$ o $f(x) = \sqrt{1+x^3}$

Solución

```
regla_simpson <- function(f, a, b, n) {
  # Verificamos que n sea par
  if (bitwAnd(n,1)) {
    return("n debe ser par")
  }
  h = (b-a)/n
  suma = 0
  x0 <- a
  suma <- suma + f(x0)
  for (i in seq(1, n-1, 1)) {
    xi <- a + i * h
    if (bitwAnd(i,1)) {
      suma <- suma + (4 * f(xi))
    } else {
      suma <- suma + (2 * f(xi))
    }
  }
  xn <- a + n * h
  suma <- suma + f(xn)
  return((h/3) * suma)
}
```

Para la implementación en R para la regla de simpson primero verificamos que n sea par para poder realizar el método, si no es par regresamos un mensaje al usuario. Una vez que sabemos que n es par proseguimos a la realización del método.
Para el metodo asignamos las variables h y xi con la correspondiente definición del método,

h sera el intervalo y xi sera el valor sumado de h por cada iteración, iteraremos 1 hasta n-1 y verificaremos si el valor de i es par o impar, dependiendo del valor multiplicaremos por 4 o 6 al valor de la función que devuelve y sumaremos el resultado en cada iteración. Una vez acabe las iteraciones sumaremos el ultimo valor que seria xn y la evaluaremos en la función, con esto tendríamos la suma de todos los términos, ahora solo nos faltaría multiplicarlo por $\frac{h}{3}$ y regresamos su valor.

Podemos observar que cuando n es muy grande la aproximación será mayor, ya que vamos sumando más porciones bajo la curva. Para probar nuestro método utilizaremos estas 4 funciones.

```
1 cuadrado <- function(x) x^2
2 seno <- function(x) sin(x)
3 e <- function(x) exp(-(x^2))
4 raiz <- function(x) sqrt(1+x^3)
```

Salida

```
> regla_simpson(cuadrado, 0, 150, 10000)
[1] 1125000
> regla_simpson(seno, 0, (3 * pi) /4, 8)
[1] 1.707179
> regla_simpson(e, 0, 100, 1000)
[1] 0.8862269
> regla_simpson(raiz, 0, 100, 10)
[1] 40008.12
```

Por último, podemos concluir que dependiendo de la función el valor de n puede variar para tener una mejor aproximación, en el ultimo ejemplo n tuvo un valor de 10 para obtener valores muy próximos. Si nosotros queremos obtener mejores aproximaciones utilizaremos una n más grande en la función que deseamos aproximar.