Métodos iterativos de proyección unidimensional - Comparación entre los métodos de descenso

Indicaciones

Tarea: Comparación de Métodos Iterativos de Proyección Unidimensional

En esta tarea se compararán **tres métodos iterativos** para la solución del sistema lineal (Ax = b), basados en la **Sección 5.3** del libro de Yousef Saad, *Iterative Methods for Sparse Linear Systems*:

- **5.3.1 – Steepest Descent**
- **5.3.2 – Minimal Residual (MR) Iteration**
- **5.3.3 – Residual Norm Steepest Descent**

Parte 1: Matrices Definidas Positivas

Seleccionar una familia de **matrices simétricas definidas positivas**, tales como:

- Matrices de **Hilbert**
- Matrices aleatorias **simétricas positivas definidas**
- Matrices **diagonales dominantes** o **laplacianas**

Comparar los tres métodos bajo condiciones uniformes:

- **Criterio de parada**: $\frac{\|Ax_k - b\|}{\|b\|} < 10^{-6}$
- **Parámetros comunes**:
  - Misma **tolerancia**
  - Mismo **número máximo de iteraciones**
- **Métricas de comparación**:
  - Número de **iteraciones hasta convergencia**
  - **Tiempo de cómputo** (puedes usar `@btime`, `@benchmark`, etc.)
  - **Precisión numérica** con `Float16`, `Float32`, `Float64`

Parte 2: Matrices No Definidas Positivas

Seleccionar una familia de **matrices invertibles pero no positivas definidas**, como por ejemplo:

- Matrices **no simétricas aleatorias**
- **Perturbaciones** de matrices diagonales dominantes
- Matrices **con espectros conocidos** no positivos

Comparar solo los métodos **MR** y **Residual Norm Steepest Descent**, usando los mismos criterios de la Parte 1.

Set Up

```
begin
    using LinearAlgebra
    using BenchmarkTools
    using Plots
    using PrettyTables
end
```

Implementación de algoritmos

Descenso más empinado (Steepest Descent)

Código

```
dme (generic function with 1 method)
```

```
function dme(A,x,b,M,tol)
    r=b-A*x
    p=A*r
    nr0=norm(r)
    for i=1:M
        nr2=dot(r,r)
        nr=sqrt(nr2)
        println("Norma r_",i," =",nr,".. ",sqrt(r'*A*r))
        if ( nr/nr0< tol)  # necesita garatizar un buen residuo inicial
            break
        end
        α=nr2/dot(r,p)
        x=x+α*r
        r=r-α*p
        p=A*r
    end
    return x
end
```

Ejemplo

```
begin
    A1 = randn(5,5)
    A = A1 * A1'
    x = zeros(5)
    b = randn(5)
    display(A)
    display(b)
end
```

```
5×5 Matrix{Float64}:
  4.76868    -0.0940934  -0.094215   0.592178    -1.72301
 -0.0940934   6.92247    -3.68209    3.0019      -1.54428
 -0.094215   -3.68209     7.18985    1.16888      4.38841
  0.592178    3.0019      1.16888    4.93225     -0.00376296
 -1.72301    -1.54428     4.38841   -0.00376296   6.15034
5-element Vector{Float64}:
 -0.01927677402876484
  1.1191350903531827
  2.0760619090064036
  0.28687082496319694
 -1.34412589505847
```

Calculemos el valor de x utilizando el método con una tolerancia de $\epsilon = 10^{-6}$ en 10000 iteraciones.

```
begin
    ϵ = 10^-6
    x1 = dme(A,x,b,10000,ϵ)
    display(x1)
    display(A*x1)
end
```

```
Norma r_1 =2.729805356886438.. 4.178627601938818
Norma r_2 =3.0985129808777008.. 7.283007907717336
Norma r_3 =1.7383930677921298.. 2.9902891351200394
Norma r_4 =2.067225359619757.. 5.150880151260303
Norma r_5 =1.3783697103329315.. 3.285191404156398
Norma r_6 =1.4645552459377278.. 4.204727283730573
Norma r_7 =1.1252634629802687.. 2.732023617496158
Norma r_8 =1.2157718945359803.. 3.4894270963370304
Norma r_9 =0.9399784486331079.. 2.2821903283130758
Norma r_10 =1.0155957217296596.. 2.914894479888122
Norma r_11 =0.7852135987424707.. 1.9064340128709276
Norma r_12 =0.8483807036340507.. 2.434965189601393
Norma r_13 =0.6559303595372633.. 1.592544944810965
Norma r_14 =0.7086971764989678.. 2.0340549322237944
Norma r_15 =0.5479332473421265.. 1.330336842043871
Norma r_16 =0.5920121545397181.. 1.6991534364592362
Norma r_17 =0.4577175598908837.. 1.111300575607416
Norma r_18 =0.49453899746014146.. 1.4193925421126221
Norma r_19 =0.3823556348586115.. 0.9283280222087889
Norma r_20 =0.41311452500064316.. 1.1856935021001545
Norma r_21 =0.3194018414888059.. 0.775481391580747
Norma r_22 =0.34509636579631076.. 0.9904723599885505
Norma r_23 =0.2668132153581927.. 0.647800534191066
Norma r_24 =0.28827720759979913.. 0.8273938367429526
Norma r_25 =0.22288316046657963.. 0.5411419753650575
Norma r_26 =0.24081316599735042.. 0.6911657394342676
Norma r_27 =0.1861860671065249.. 0.452044452028407
Norma r_28 =0.20116394702345974.. 0.5773672200028896
Norma r_29 =0.15553104824981356.. 0.377616588459677
Norma r_30 =0.16804286183631784.. 0.48230531074462174
Norma r_31 =0.12992329311003883.. 0.3154430659199153
Norma r_32 =0.1403750713384391.. 0.40289508083140735
```

Residuo Mínimo (Minimal Residual Iteration)

Código

```
mr (generic function with 1 method)
```

```
function mr(A,x,b,M,tol)
    r=b-A*x
    p=A*r
    nr0=norm(r)
    for i=1:M
        nr2=dot(r,r)
        nr=sqrt(nr2)
        println("Norma r_",i," =",nr)
        if ( nr/nr0< tol)  # necesita garatizar un buen residuo inicial
            break
        end
        α=nr2/dot(p,p)
        x=x+α*r
        r=r-α*p
        p=A*r
    end
    return x
end
```

Ejemplo

Utilizando la matriz y los vectores definidos anteriormente.

```
begin
    x2 = mr(A,x,b,10000,ϵ)
    display(x2)
    display(A*x2)
end
```

```
Norma r_1 =2.729805356886438
Norma r_2 =2.2946610235638674
Norma r_3 =1.8299333927802746
Norma r_4 =1.5207430619471456
Norma r_5 =1.3143720204288438
Norma r_6 =1.006674427064016
Norma r_7 =0.877018773486456
Norma r_8 =0.7620526322090246
Norma r_9 =0.6605187442348911
Norma r_10 =0.5711709658725479
Norma r_11 =0.4927957179864965
Norma r_12 =0.42423211563751
Norma r_13 =0.36438812561728434
Norma r_14 =0.312251510526328
Norma r_15 =0.26689547315495016
Norma r_16 =0.22747962315269687
Norma r_17 =0.19324715710942542
Norma r_18 =0.16351906387596118
Norma r_19 =0.13768580594294716
Norma r_20 =0.11519622836833518
Norma r_21 =0.09554205320390925
Norma r_22 =0.07823305942590406
Norma r_23 =0.06274930859131687
Norma r_24 =0.04843348093532372
Norma r_25 =0.03429448088569748
Norma r_26 =0.02052310704256488
Norma r_27 =0.013635347030114863
Norma r_28 =0.011193472661043426
Norma r_29 =0.008318657554698527
Norma r_30 =0.006948386708058252
Norma r_31 =0.005789252097300729
Norma r_32 =0.005307432571786521
```

Norma residual del descenso más empinado (Residual Norm Steepest Descent)

El algoritmo es:

1. $$r = b - Ax$$

Hasta convergencia, hacer

1. $$v = A^T r$$
2. $$\alpha = \|v\|^2 / \|(Av)\|^2$$
3. $$x = x + \alpha r$$
4. $$r = r - \alpha p$$

Código

```
dpnr (generic function with 1 method)
```

```
function dpnr(A,x,b,M,tol)
    r = b - A*x
    nr0 = norm(r)
    for i = 1:M
        v = A' * r
        α = dot(r,v) / dot(v,v)
        x = x + α * r
        r = r - α * v

        nr = norm(r)
        println("Norma r_",i," =",nr)

        if nr/nr0 < tol
            break
        end
    end

    return x
end
```

Ejemplo

Utilizando la matriz y los vectores definidos anteriormente.

```
begin
    x3 = dpnr(A,x,b,10000,ϵ)
    display(x3)
    display(A*x3)
end
```

```
Norma r_1 =2.048280567070545
Norma r_2 =1.699761142381453
Norma r_3 =1.4532985293765874
Norma r_4 =1.2957678227961245
Norma r_5 =1.1762557864297654
Norma r_6 =1.0749271239424723
Norma r_7 =0.9824815940707537
Norma r_8 =0.8980416029984315
Norma r_9 =0.8208881519867658
Norma r_10 =0.7503748011303333
Norma r_11 =0.685924915658878
Norma r_12 =0.6270132043953438
Norma r_13 =0.5731626353028997
Norma r_14 =0.5239375456195365
Norma r_15 =0.4789403807914881
Norma r_16 =0.43780781730377094
Norma r_17 =0.40020788707341215
Norma r_18 =0.3658371515560566
Norma r_19 =0.33441826570466016
Norma r_20 =0.30569770772686833
Norma r_21 =0.2794437362045499
Norma r_22 =0.25544451310519023
Norma r_23 =0.23350639512162433
Norma r_24 =0.2134523696521482
Norma r_25 =0.19512062667092459
Norma r_26 =0.1783632527863382
Norma r_27 =0.16304503780170587
Norma r_28 =0.149042383669217
Norma r_29 =0.1362423072212273
Norma r_30 =0.12454152852668647
Norma r_31 =0.1138456375581786
Norma r_32 =0.10406833242243281
```

Comparaciones entre los métodos

Norma del error absoluto

Evaluemos el error de las ejecuciones del ejemplo en los tres métodos anteriores.

```
begin
    er1 = norm(x1 - A*x)
    er2 = norm(x2 - A*x)
    er3 = norm(x3 - A*x)

    println("Norma del error utilizando dme: ", er1)
    println("Norma del error utilizando mr: ", er2)
    println("Norma del error utilizando dpnr: ", er3)
end
```

```
Norma del error utilizando dme: 3.6693799173486297
Norma del error utilizando mr: 3.6693832501611667
Norma del error utilizando dpnr: 3.6693791154274416
```

En matrices simétricas definidas positivas

Evaluemos los métodos para realizar el análisis comparativo.

Número de iteraciones hasta convergencia

Definamos 3 matrices de tamaños distintos para ver los resultados.

```
begin
    B1 = randn(10,10)
    B1 = B1*B1'
    B2 = randn(50,50)
    B2 = B2*B2'
    B3 = randn(100,100)
    B3 = B3*B3'

    display(B1)
end
```

```
10×10 Matrix{Float64}:
  4.61076    2.687      0.711519  …   1.40025    0.168241  -2.31437
  2.687      8.01505   -1.11291      -3.88149   -3.13267    0.677029
  0.711519  -1.11291    2.68903       0.237605  -0.896109  -2.46159
 -0.888281   0.453618   0.13763       0.426725   1.41994    4.48346
 -2.81178    3.21462   -1.22818      -6.55677    2.29007    2.98089
 -0.1574     1.05026    1.34889   …  -1.01147   -0.144246  -2.98653
  1.20512    2.13124    2.04104      -2.41339   -3.52785   -0.157752
  1.40025   -3.88149    0.237605      7.03176    1.90577    0.141876
  0.168241  -3.13267   -0.896109      1.90577   11.7679     2.87582
 -2.31437    0.677029  -2.46159       0.141876   2.87582   10.6967
```

Para ver el comportamiento de los métodos vamos a modificarlas para obtener la norma de cada iteración.

```
dmeH (generic function with 1 method)
```

```
function dmeH(A, x, b, M, tol)
    r = b - A * x
    p = A * r
    nr0 = norm(r)
    history = [nr0]

    for i in 1:M
        nr2 = dot(r, r)
        α = nr2 / dot(r, p)
        x = x + α * r
        r = r - α * p
        p = A * r
        nr = norm(r)
        push!(history, nr)
        if nr / nr0 < tol
            break
        end
    end

    return x, history
end
```

```
mrH (generic function with 1 method)
```

```
function mrH(A,x,b,M,tol)
    r=b-A*x
    p=A*r
    nr0=norm(r)
    history = [nr0]
    for i=1:M
        nr2=dot(r,r)
        α=nr2/dot(p,p)
        x=x+α*r
        r=r-α*p
        p=A*r
        nr=sqrt(nr2)
        push!(history, nr)
        if ( nr/nr0< tol)
            break
        end
    end
    return x, history
end
```

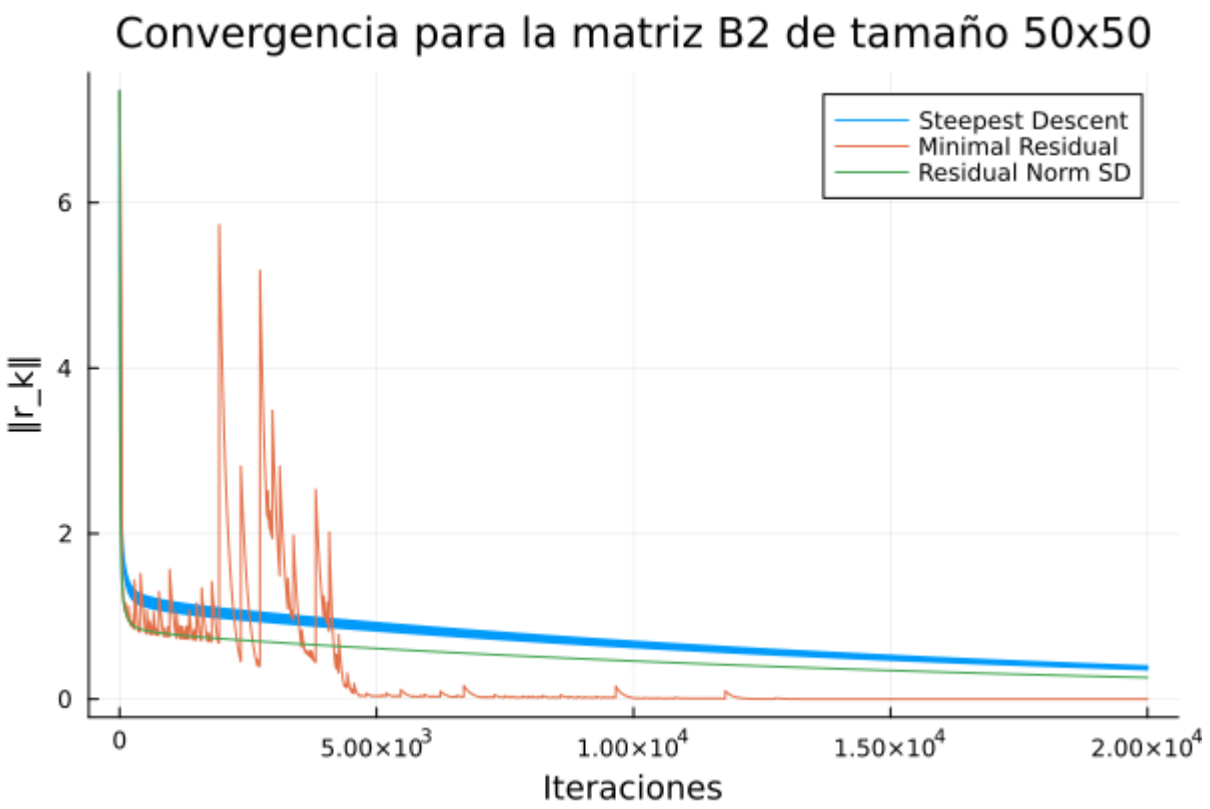
```
dmpH (generic function with 1 method)
```

```
function dmpH(A, x, b, M, tol)
    r = b - A * x
    nr0 = norm(r)
    history = [nr0]

    for i in 1:M
        Ar = A * r
        α = dot(r, Ar) / dot(Ar, Ar)
        x = x + α * r
        r = r - α * Ar
        nr = norm(r)
        push!(history, nr)
        if nr / nr0 < tol
            break
        end
    end

    return x, history
end
```

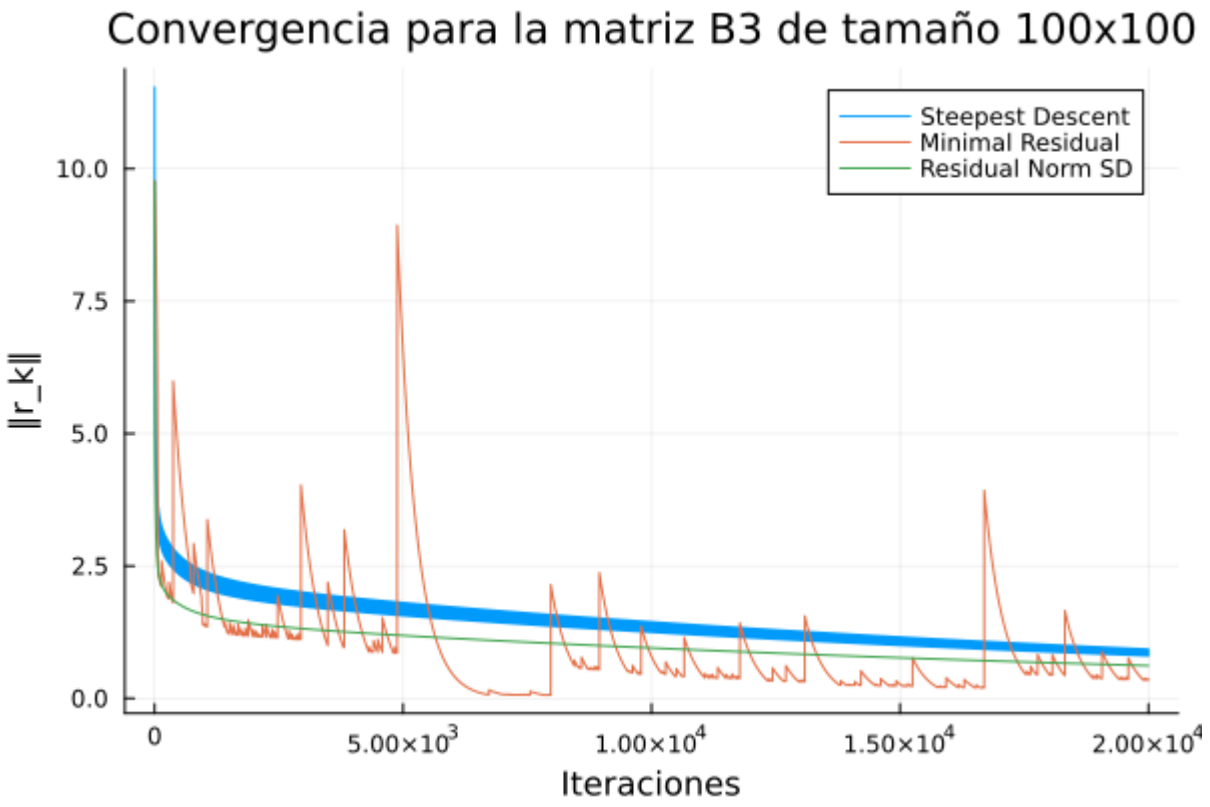
```
convergence_experiment (generic function with 1 method)
```

```
function convergence_experiment(A, name)
    b = randn(size(A,1))
    x0 = zeros(size(A,1))
    tol = 1e-8
    M = 20000

    _, hist_sd   = dmeH(A, x0, b, M, tol)
    _, hist_rnsd = dmpH(A, x0, b, M, tol)
    _, hist_minres = mrH(A, x0, b, M, tol)

    plot(1:length(hist_sd), hist_sd; label="Steepest Descent")
    plot!(1:length(hist_minres), hist_minres; label="Minimal Residual")
    plot!(1:length(hist_rnsd), hist_rnsd; label="Residual Norm SD")
    xlabel!("Iteraciones")
    ylabel!("‖r_k‖")
    title!(
        "Convergencia para la matriz $name de tamaño "
        * string(size(A,1)) * "x" * string(size(A,2)))
end
```


```
convergence_experiment(B1, "B1")
```




```
1 convergence_experiment(B2, "B2")
```




```
1 convergence_experiment(B3, "B3")
```

Resultados

Al comparar las gráficas con distintos tamaños se puede observar que el método de Mínimo Residuo converge más rápidamente que los otros dos. Sin embargo, en algunas iteraciones dicha norma no es tan estable como los otros dos métodos, esto es observable en las oscilaciones presentadas en los tres experimentos.

Tiempo de Cómputo

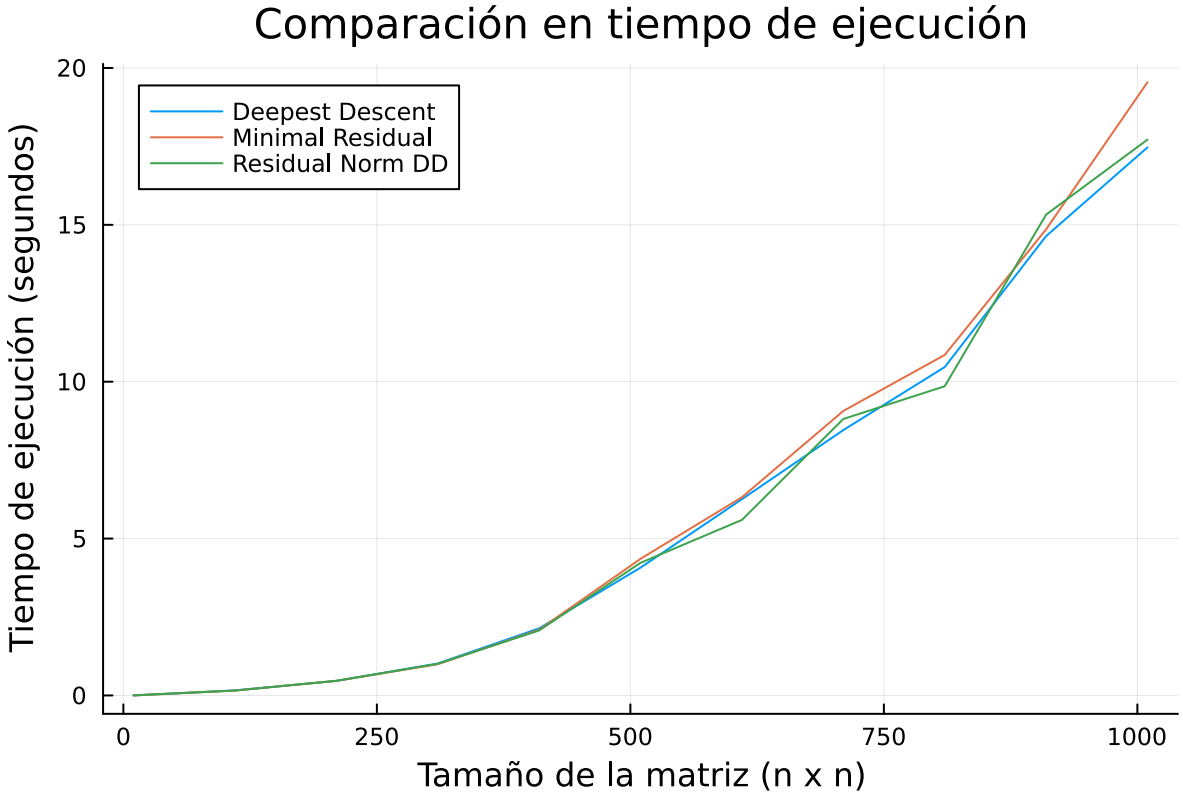
Comparemos los tres métodos con matrices de tamaño 10×10 a tamaño 1000×1000.

```
benchmark_metodos (generic function with 1 method)
```

```
function benchmark_metodos(ns)

    times_dme = Float64[]
    times_mr = Float64[]
    times_dmp = Float64[]

    for n in ns
        A = randn(n,n)
        A = A*A'

        x0 = zeros(n)
        b = randn(n)
        tol = 1e-12
        M = 50000

        t_dme = @belapsed dmeH($A, $x0, $b, $M, $tol)
        t_mr  = @belapsed mrH($A, $x0, $b, $M, $tol)
        t_dmp = @belapsed dmpH($A, $x0, $b, $M, $tol)

        push!(times_dme,t_dme)
        push!(times_mr,t_mr)
        push!(times_dmp,t_dmp)
    end
    return times_dme, times_mr, times_dmp
end
```

```
([0.00145663, 0.160666, 0.463701, 1.01774, 2.13825, 4.07352, 6.25471, 8.45762, 10.4669,
```

```
begin
    ns = 10:100:1010
    times_dme, times_mr, times_dmp = benchmark_metodos(ns)
end
```




```
begin
    plot(ns,times_dme, label="Deepest Descent")
    plot!(ns,times_mr, label="Minimal Residual")
    plot!(ns,times_dmp, label="Residual Norm DD")
    xlabel!("Tamaño de la matriz (n x n)")
    ylabel!("Tiempo de ejecución (segundos)")
    title!("Comparación en tiempo de ejecución")
end
```

Resultados

En los experimentos se logra evidenciar que el tiempo de cómputo entre los tres métodos es muy parecido; no parece haber indicios de que una sea más rápida que las otras.

Precisión utilizando distintos formatos

Ahora veamos el comportamiento de los algoritmos utilizando distintos formatos de punto flotante.

```
formato (generic function with 1 method)
```

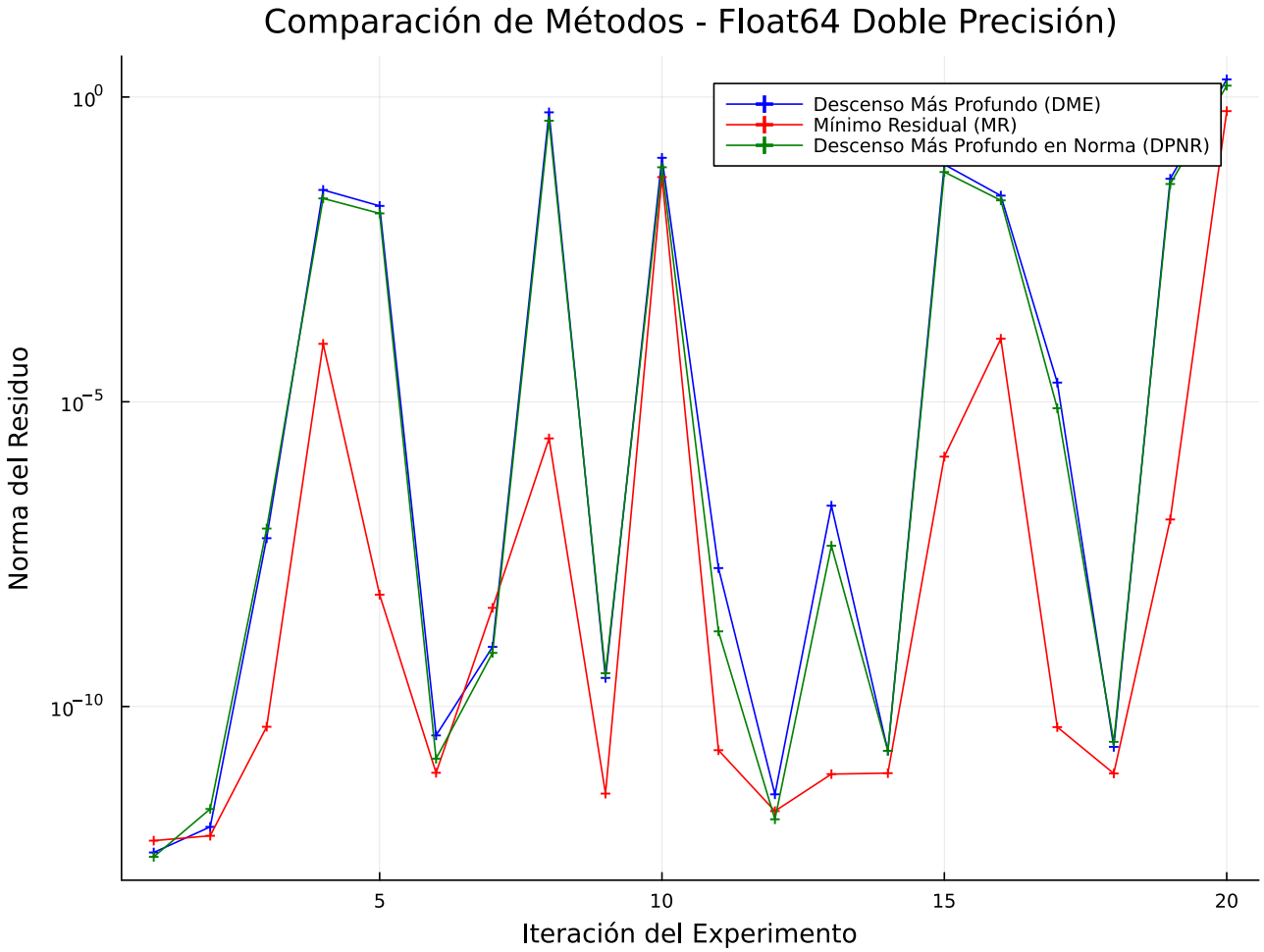
```
function formato(num::Int, tipo::Type{T}) where T <: AbstractFloat
    # Inicializar arreglos para almacenar resultados
    residuos_dme = T[]
    residuos_mr = T[]
    residuos_dpnr = T[]

    # Definir tolerancia según el tipo de datos
    tolerancia = if tipo == Float64
        1e-14
    elseif tipo == Float32
        1e-7
    elseif tipo == Float16
        1e-4
    else
        1e-8  # valor por defecto
    end

    # Número máximo de iteraciones
    M = 50000

    # Ejecutar experimento num veces
    for _ in 1:num
        # Generar matriz A simétrica definida positiva
        A_temp = randn(20, 20)
        A = tipo.(A_temp * A_temp')

        # Vector inicial y vector b
        x0 = zeros(tipo, 20)
        b = tipo.(randn(20))

        # Ejecutar cada método
        x_dme,h1 = dmeH(A, x0, b, M, tolerancia)
        x_mr,h2 = mrH(A, x0, b, M, tolerancia)
        x_dpnr,h3 = dmpH(A, x0, b, M, tolerancia)

        # Calcular residuos y almacenar
        push!(residuos_dme, norm(b - A * x_dme))
        push!(residuos_mr, norm(b - A * x_mr))
        push!(residuos_dpnr, norm(b - A * x_dpnr))
    end
    return residuos_dme,residuos_mr,residuos_dpnr
end
```

```
([60.34, 333.5, 96.4, 931.5, 21.0, 15.04, 1.676e3, 1.613e3, 104.06,  more ,336.0], [Na
```

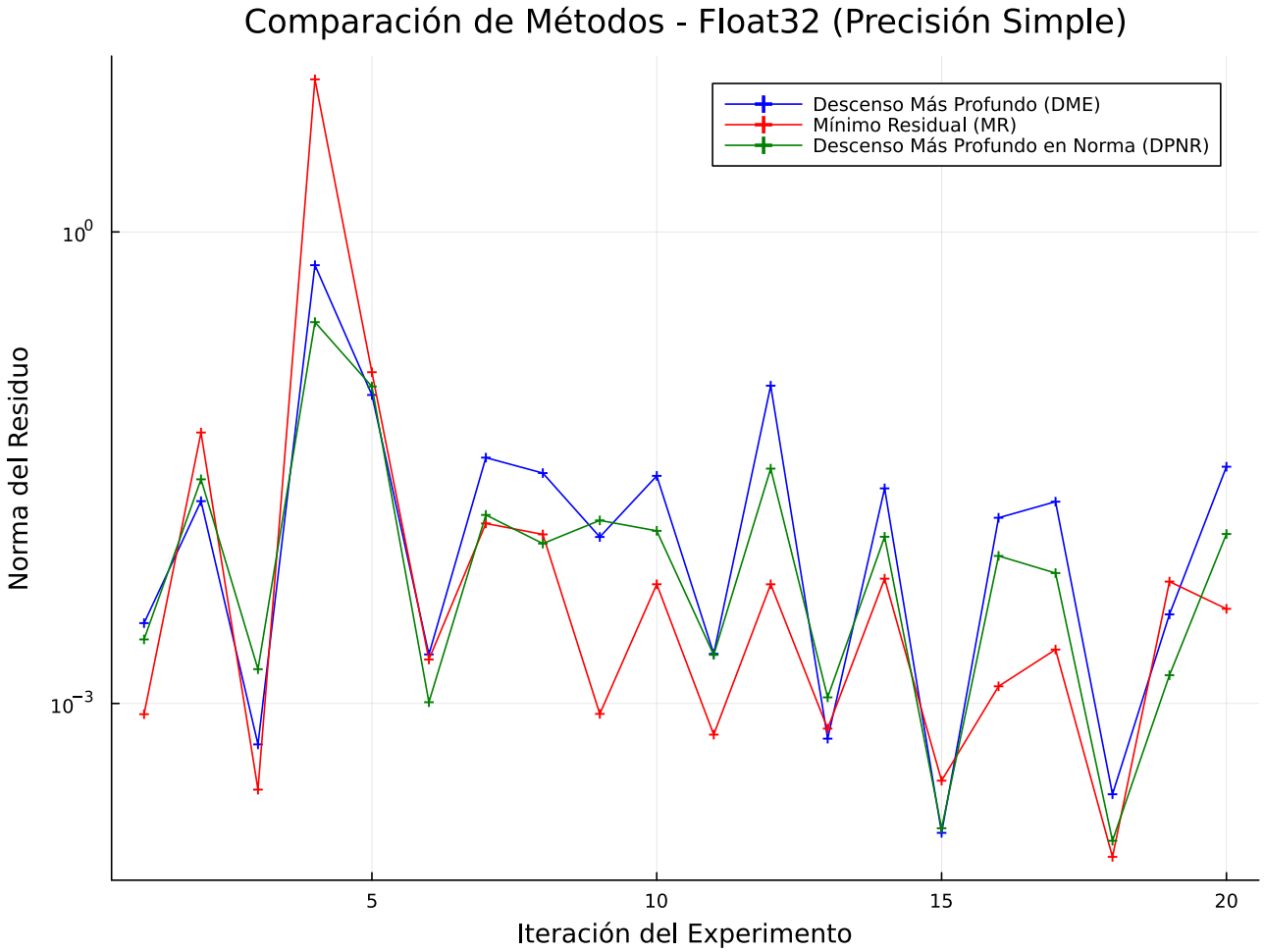
```
begin
    resultados_fl64 = formato(20,Float64)
    resultados_fl32 = formato(20,Float32)
    resultados_fl16 = formato(20,Float16)
end
```

```
graficar_formato (generic function with 1 method)
```

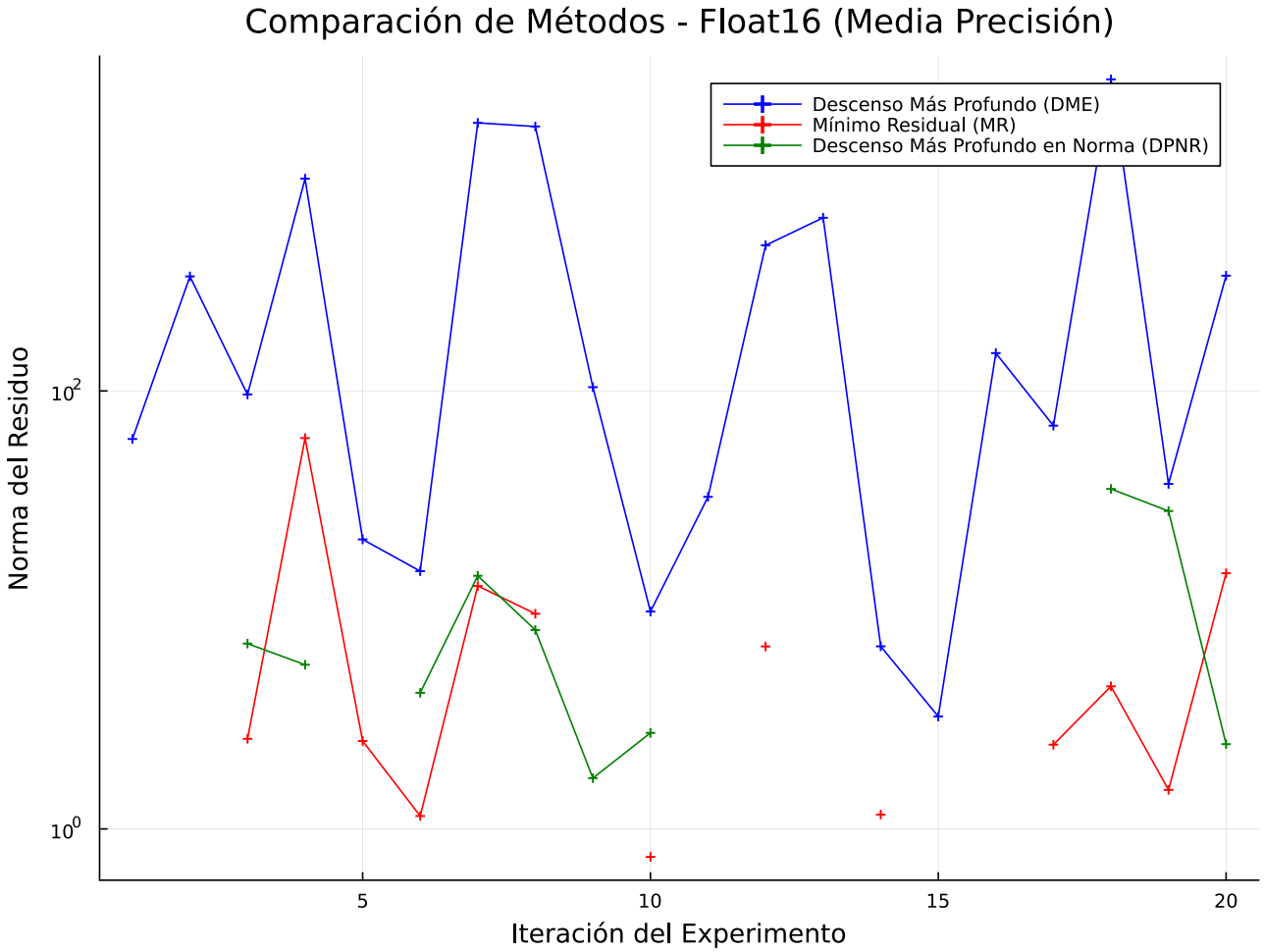
```
function graficar_formato(residuos_dme, residuos_mr, residuos_dpnr,
tipo::Type{T};
                          titulo_personalizado="") where T <: AbstractFloat

    # Crear nombre del formato para el título
    nombre_formato = string(tipo)
    titulo = isempty(titulo_personalizado) ?
             "Comparación de Métodos - $nombre_formato" :
             titulo_personalizado

    # Crear la gráfica
    p =plot(title=titulo,
             xlabel="Iteración del Experimento",
             ylabel="Norma del Residuo",
             legend=:topright,
             yscale=:log10,  # Escala logarítmica para mejor visualización
             size=(800, 600),
             dpi=300)

    # Añadir cada método a la gráfica
    plot!(p, 1:length(residuos_dme), residuos_dme,
          label="Descenso Más Profundo (DME)",
          marker=:+,
          markersize=3,
          color=:blue)

    plot!(p, 1:length(residuos_mr), residuos_mr,
          label="Mínimo Residual (MR)",
          marker=:+,
          markersize=3,
          color=:red)

    plot!(p, 1:length(residuos_dpnr), residuos_dpnr,
          label="Descenso Más Profundo en Norma (DPNR)",
          marker=:+,
          markersize=3,
          color=:green)
end
```




```
begin
    fl64_dme, fl64_mr, fl64_dpnr = resultados_fl64
    fl32_dme, fl32_mr, fl32_dpnr = resultados_fl32
    fl16_dme, fl16_mr, fl16_dpnr = resultados_fl16

    graficar_formato(fl64_dme, fl64_mr, fl64_dpnr, Float64,
                     titulo_personalizado="Comparación de Métodos - Float64 Doble Precisión)")
end
```




```
begin
    graficar_formato(fl32_dme, fl32_mr, fl32_dpnr, Float32,
                     titulo_personalizado="Comparación de Métodos - Float32
(Precisión Simple)")
end
```




```
begin
    graficar_formato(fl16_dme, fl16_mr, fl16_dpnr, Float16,
                     titulo_personalizado="Comparación de Métodos - Float16
(Media Precisión)")
end
```

De los cuales podemos identificar que, aunque los comportamientos son similares, claramente, cuando utilizamos formatos más simples obtenemos resultados más pobres. Esto podemos corroborarlo revisando el promedio de los resultados.

```
begin
    r1 = mean(reduce(vcat, resultados_fl64))
    r2 = mean(reduce(vcat, resultados_fl32))
    r3 = mean(filter(!isnan, (reduce(vcat, resultados_fl16))))

    println("Promedio en los resultados utilizando precisión doble: ",r1)
    println("Promedio en los resultados utilizando precisión simple: ",r2)
    println("Promedio en los resultados utilizando precisión media: ",r3)

end
```

```
Promedio en los resultados utilizando precisión doble: 0.093441676695821
43
Promedio en los resultados utilizando precisión simple: 0.18606324
Promedio en los resultados utilizando precisión media: 219.9
```

Resultados

En los experimentos se logra evidenciar que el error en el uso de cada uno de los formatos se hace menor al utilizar formatos más precisos. Además, utilizando el formato de precisión media muchos valores tienen a desbordar la pila de memoria.

En matrices no definidas positivas

En este caso solo evaluaremos los métodos de Residuos Mínimos y Descenso más Empinado de la Norma Residual:

Número de iteraciones hasta convergencia

Definamos 3 matrices de tamaños distintos para ver los resultados. Para esto creamos matrices diagonales dominantes con perturbaciones:

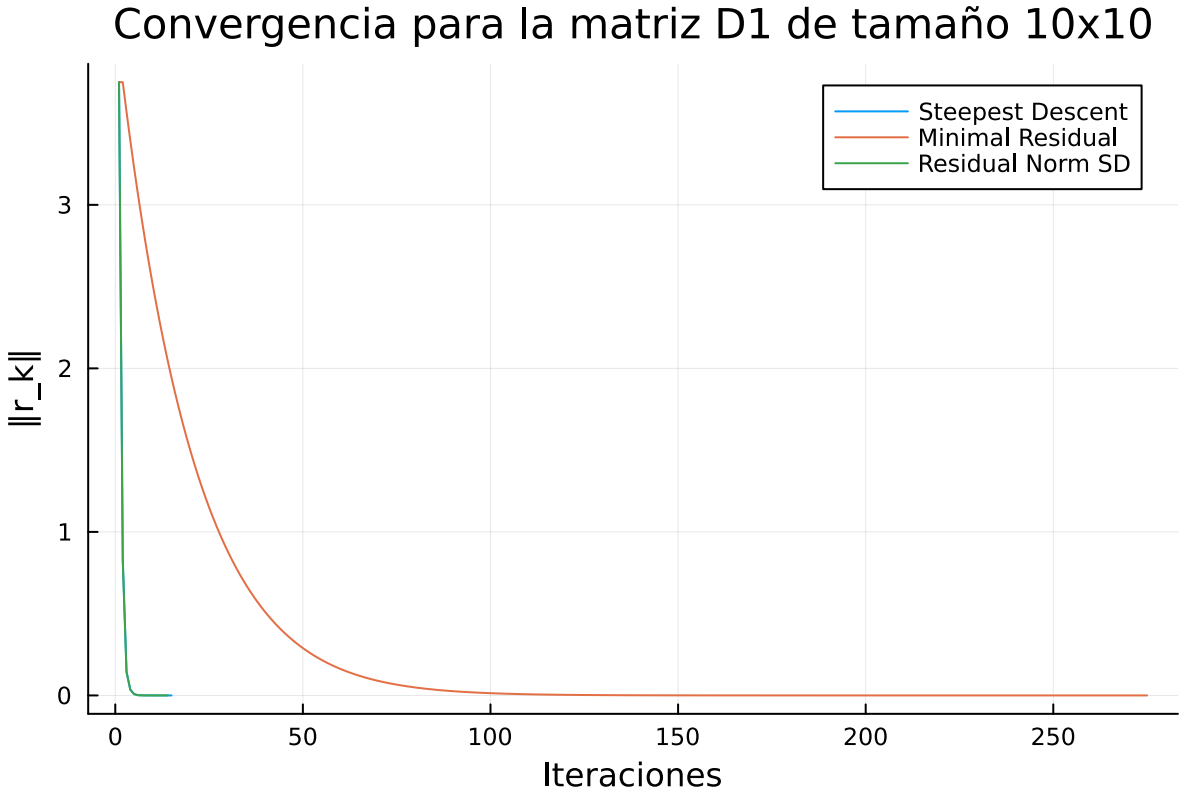
```
generar_matriz_perturbada (generic function with 1 method)
```

```
function generar_matriz_perturbada(n::Int; escala_perturbación::Float64=0.2)
    A = randn(n, n)

    # Hacer diagonalmente dominante
    for i in 1:n
        A[i, i] = 2.0 * sum(abs.(A[i, :])) + 1.0
    end

    # Añadir perturbación pequeña para romper simetría
    perturbacion = escala_perturbación * randn(n, n)
    return A + perturbacion
end

```

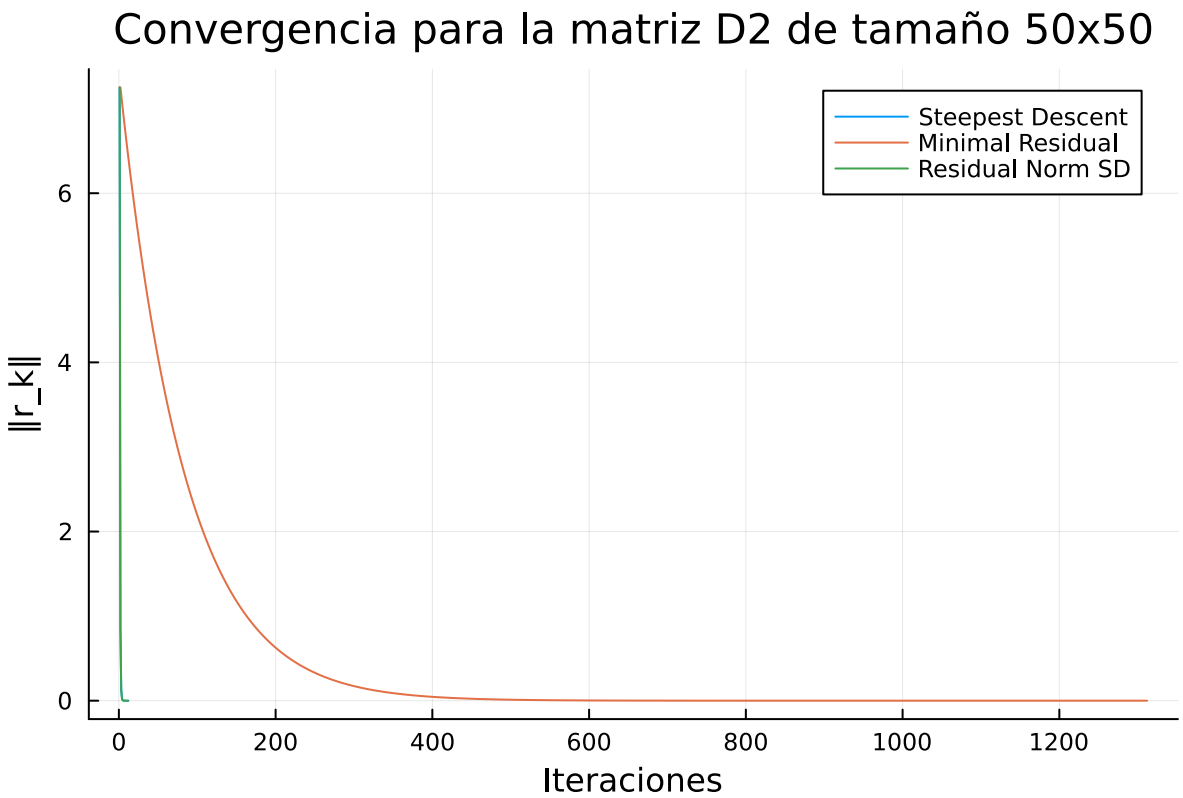
```
begin
    D1 = generar_matriz_perturbada(10)
    D2 = generar_matriz_perturbada(50)
    D3 = generar_matriz_perturbada(100)

    display(D1)
end
```

```
10×10 Matrix{Float64}:
 21.2764      1.53034     2.82142   …   0.473218   1.63908   -0.631659
 -0.0925059  13.6523      0.768928      0.907785   0.755173  -0.524277
 -0.0935287   0.0882606  17.9488       -0.919003  -0.575422   1.87968
 -1.73924     1.59159    -0.978383     -0.86093   -0.514311   1.30789
 -2.4334     -1.51973     0.235847      0.896172  -1.45177    0.660091
 -1.45303    -1.49796     1.27269   …  -1.75743    2.07316    0.892383
 -0.584829    0.0784287   0.475541      1.4088    -1.27446   -0.627822
  0.350379    0.685269   -0.784669     17.525     -0.402405   0.0475628
  1.17326     1.56668    -1.51598      -1.00897   20.4729     0.20154
 -2.58154    -0.275334   -1.93005      -0.105889   1.00066   23.8346
```

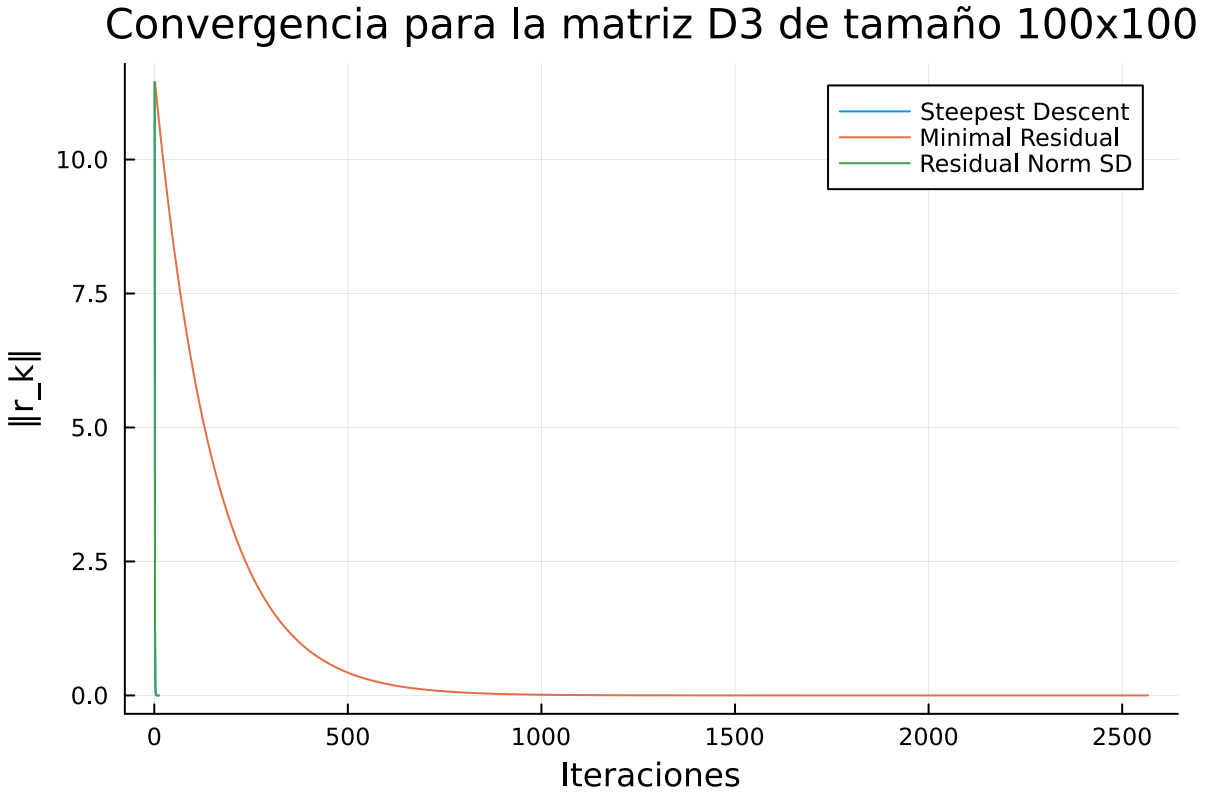
Para graficar, reutilizamos la función `convergence_experiment`




```
1 convergence_experiment(D1,"D1")
```




```
1 convergence_experiment(D2,"D2")
```




```
1 convergence_experiment(D3, "D3")
```

Resultados

Al comparar las gráficas con distintos tamaños se puede observar que el método de Descenso más empinado en la norma del residuo converge bastante más rápido que el método de Residuo Mínimo.

Tiempo de Cómputo

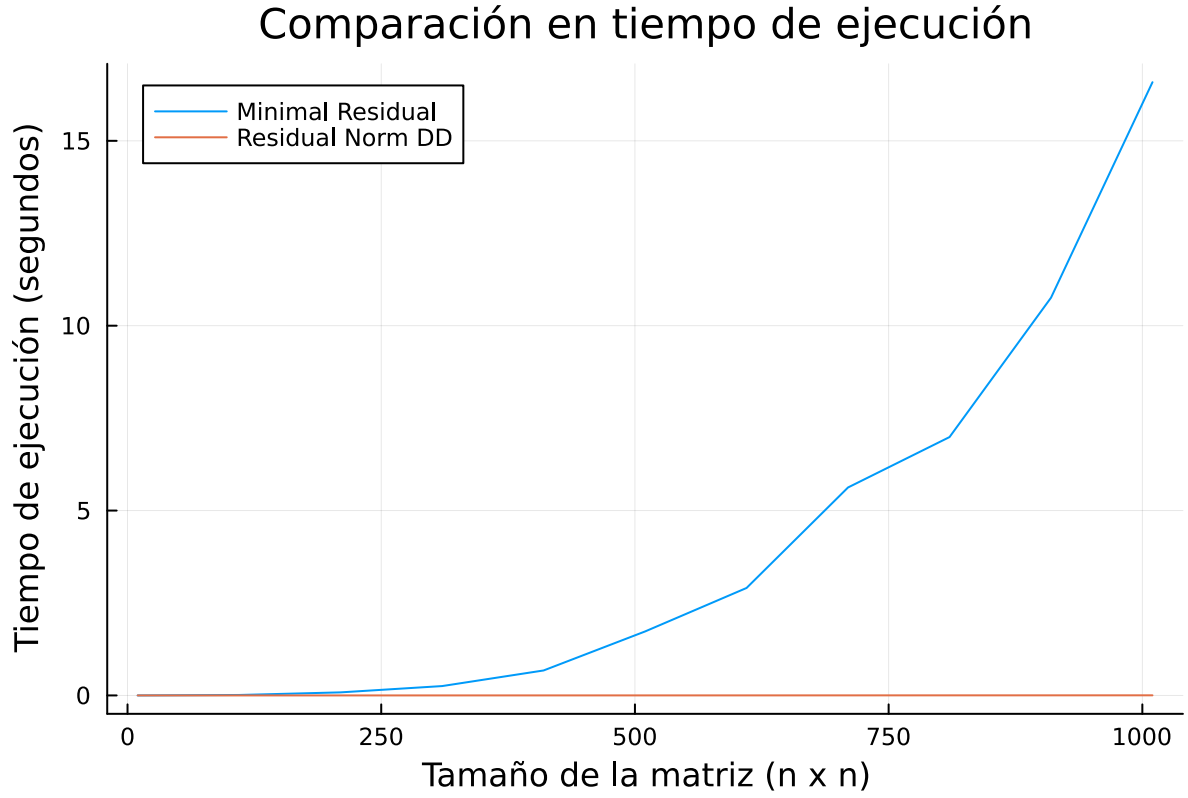
Comparemos los dos métodos con matrices de tamaño 10×10 a tamaño 1000×1000.

```
benchmark_metodos2 (generic function with 1 method)
```

```
function benchmark_metodos2(ns)

    times_mr = Float64[]
    times_dmp = Float64[]

    for n in ns
        A = randn(n,n)
        for i in 1:n
            A[i, i] = 2.0 * sum(abs.(A[i, :])) + 1.0
        end
        perturbacion = 0.2 * randn(n,n)
        A = A + perturbacion

        x0 = zeros(n)
        b = randn(n)
        tol = 1e-12
        M = 50000

        t_mr  = @belapsed mrH($A, $x0, $b, $M, $tol)
        t_dmp = @belapsed dmpH($A, $x0, $b, $M, $tol)

        push!(times_mr,t_mr)
        push!(times_dmp,t_dmp)
    end
    return times_mr, times_dmp
end
```

```
([8.2658e-5, 0.011947, 0.0828572, 0.253091, 0.675364, 1.73037, 2.90626, 5.62481, 6.987
```

```
begin
    times_mr2, times_dmp2 = benchmark_metodos2(ns)
end
```




```
begin
    plot(ns,times_mr2, label="Minimal Residual")
    plot!(ns,times_dmp2, label="Residual Norm DD")
    xlabel!("Tamaño de la matriz (n x n)")
    ylabel!("Tiempo de ejecución (segundos)")
    title!("Comparación en tiempo de ejecución")
end
```

Resultados

En los experimentos se logra evidenciar que el tiempo de cómputo del método de Residuo Mínimo crece de forma casi exponencial en relación al tiempo del método de Descenso más empinado en la norma del residuo.

Precisión utilizando distintos formatos

Ahora veamos el comportamiento de los algoritmos utilizando distintos formatos de punto flotante.

```
formato2 (generic function with 1 method)
```

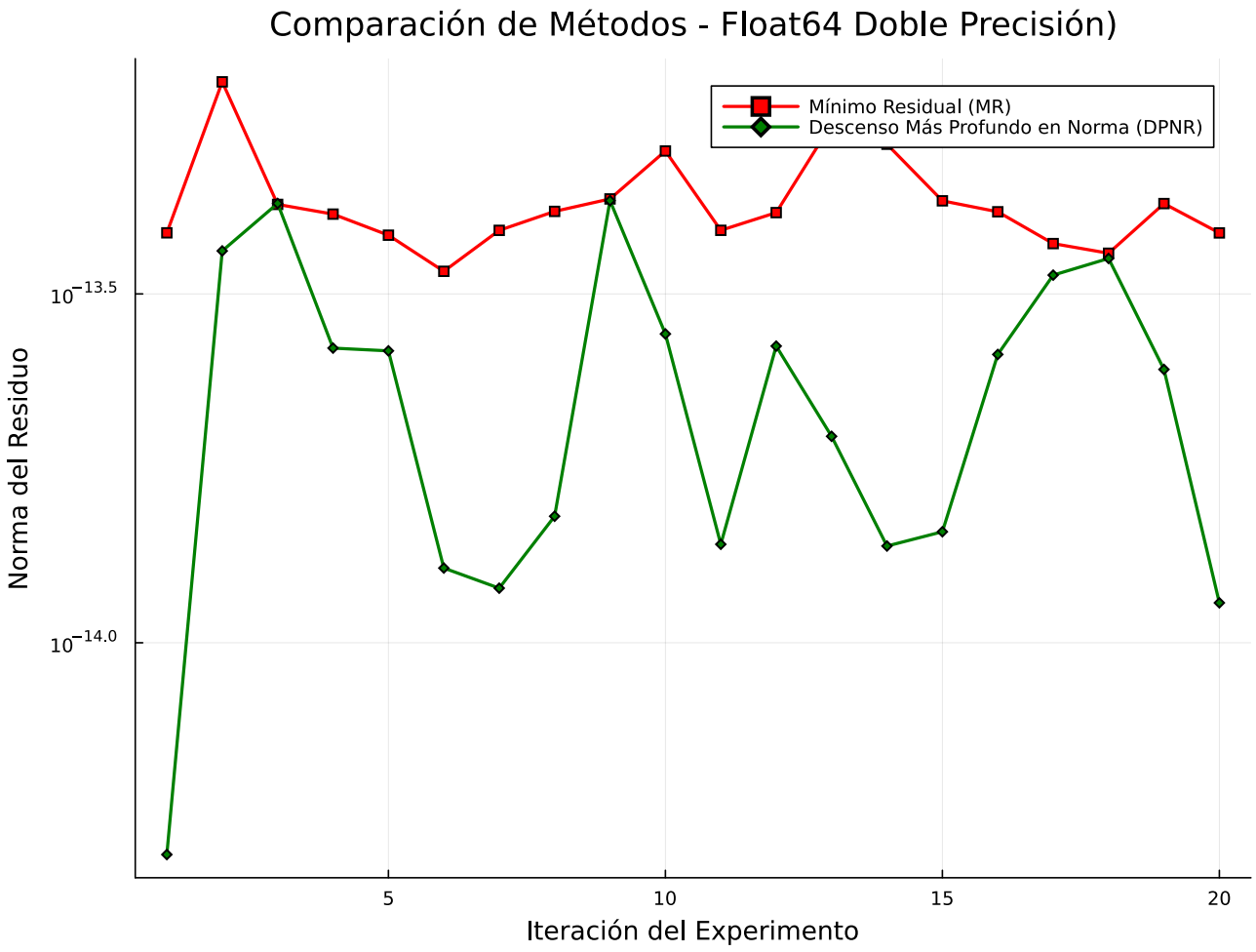
```
function formato2(num::Int, tipo::Type{T}) where T <: AbstractFloat
    # Inicializar arreglos para almacenar resultados
    residuos_mr = T[]
    residuos_dpnr = T[]

    # Definir tolerancia según el tipo de datos
    tolerancia = if tipo == Float64
        1e-14
    elseif tipo == Float32
        1e-7
    elseif tipo == Float16
        1e-4
    else
        1e-8  # valor por defecto
    end

    # Número máximo de iteraciones
    M = 50000

    # Ejecutar experimento num veces
    for _ in 1:num
        # Generar matriz A simétrica definida positiva
        A = randn(20,20)
        for i in 1:20
            A[i, i] = 2.0 * sum(abs.(A[i, :])) + 1.0
        end
        perturbacion = 0.2 * randn(20,20)
        A = A + perturbacion

        # Vector inicial y vector b
        x0 = zeros(tipo, 20)
        b = tipo.(randn(20))

        # Ejecutar cada método
        x_mr,h1 = mrH(A, x0, b, M, tolerancia)
        x_dpnr,h2 = dmpH(A, x0, b, M, tolerancia)

        # Calcular residuos y almacenar
        push!(residuos_mr, norm(b - A * x_mr))
        push!(residuos_dpnr, norm(b - A * x_dpnr))
    end
    return residuos_mr,residuos_dpnr
end
```

```
([0.0004284, 0.0003023, 0.0004463, 0.0003777, 0.0003905, 0.0006137, 0.0005097, 0.00051
```

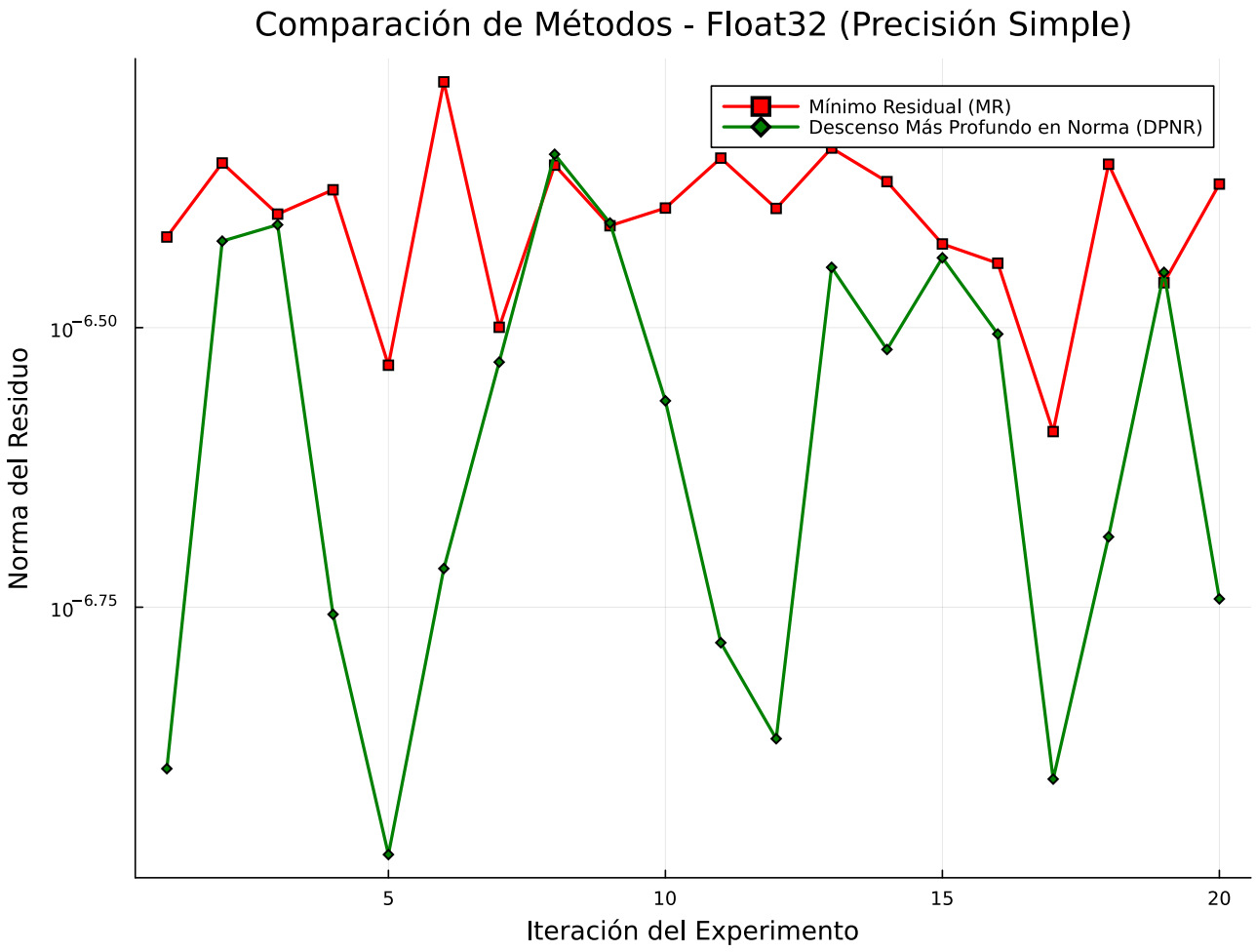
```
begin
    resultados2_fl64 = formato2(20,Float64)
    resultados2_fl32 = formato2(20,Float32)
    resultados2_fl16 = formato2(20,Float16)
end
```

```
graficar_formato2 (generic function with 1 method)
```

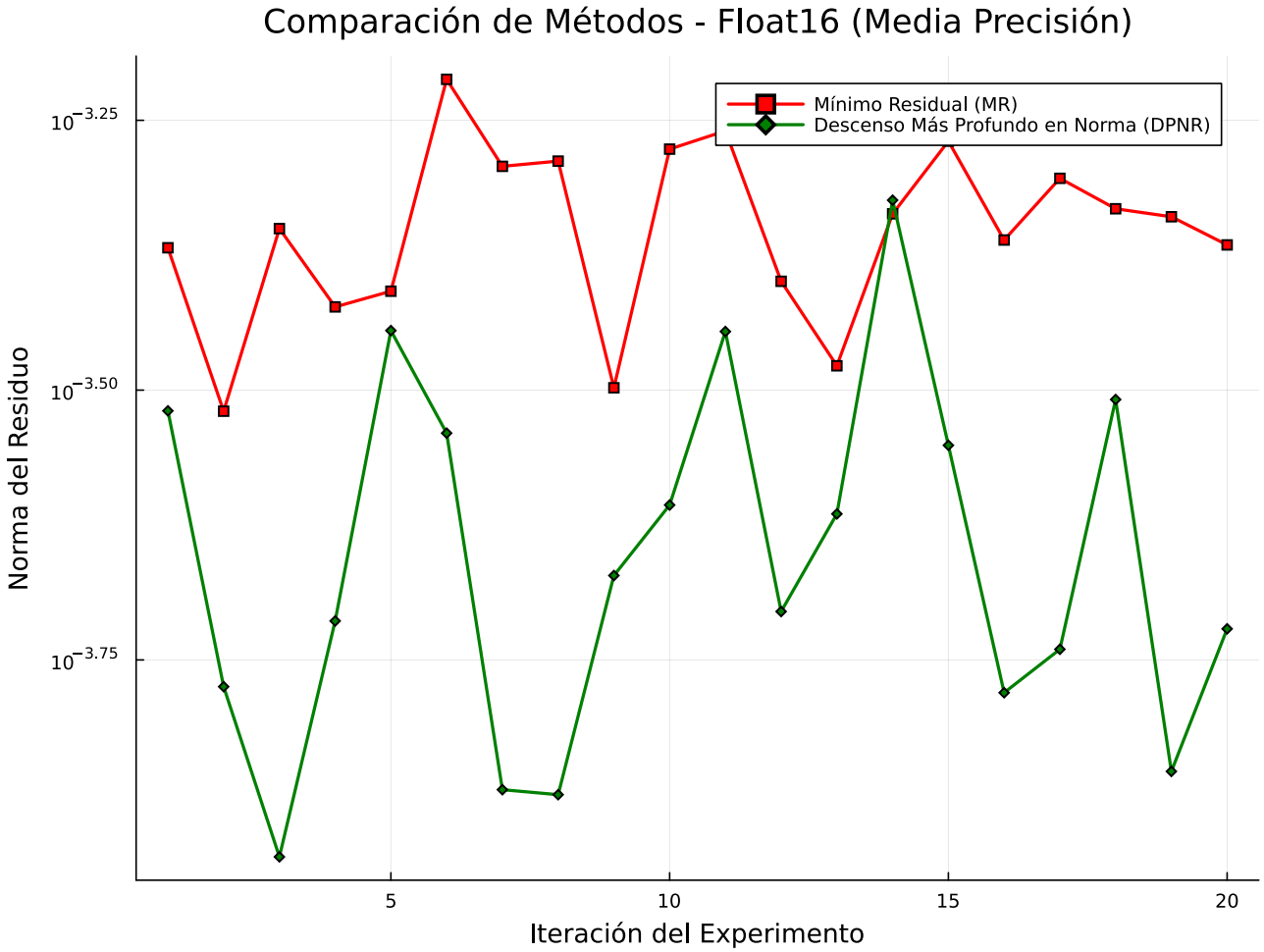
```
function graficar_formato2(residuos_mr, residuos_dpnr, tipo::Type{T};
                           titulo_personalizado="") where T <: AbstractFloat

    # Crear nombre del formato para el título
    nombre_formato = string(tipo)
    titulo = isempty(titulo_personalizado) ?
             "Comparación de Métodos - $nombre_formato" :
             titulo_personalizado

    # Crear la gráfica
    p =plot(title=titulo,
             xlabel="Iteración del Experimento",
             ylabel="Norma del Residuo",
             legend=:topright,
             yscale=:log10,  # Escala logarítmica para mejor visualización
             size=(800, 600),
             dpi=300)

    # Añadir cada método a la gráfica

    plot!(p, 1:length(residuos_mr), residuos_mr,
          label="Mínimo Residual (MR)",
          marker=:square,
          linewidth=2,
          markersize=3,
          color=:red)

    plot!(p, 1:length(residuos_dpnr), residuos_dpnr,
          label="Descenso Más Profundo en Norma (DPNR)",
          marker=:diamond,
          linewidth=2,
          markersize=3,
          color=:green)
end
```




```
begin
    fl64_mr2, fl64_dpnr2 = resultados2_fl64
    fl32_mr2, fl32_dpnr2 = resultados2_fl32
    fl16_mr2, fl16_dpnr2 = resultados2_fl16

    graficar_formato2(fl64_mr2, fl64_dpnr2, Float64,
                      titulo_personalizado="Comparación de Métodos - Float64 Doble Precisión)")
end
```




```
1 begin
2     graficar_formato2(fl32_mr2, fl32_dpnr2, Float32,
3                       titulo_personalizado="Comparación de Métodos - Float32
  (Precisión Simple)")
4 end
```




```
begin
    graficar_formato2(fl16_mr2, fl16_dpnr2, Float16,
                      titulo_personalizado="Comparación de Métodos - Float16
(Media Precisión)")
end
```

De los cuales podemos identificar que, aunque los comportamientos son similares, cuando utilizamos formatos más simples obtenemos resultados más pobres. Esto podemos corroborarlo revisando el promedio de los resultados.

```
begin
    r4 = mean(reduce(vcat, resultados2_fl64))
    r5 = mean(reduce(vcat, resultados2_fl32))
    r6 = mean(reduce(vcat, resultados2_fl16))

    println("Promedio en los resultados utilizando precisión doble: ",r4)
    println("Promedio en los resultados utilizando precisión simple: ",r5)
    println("Promedio en los resultados utilizando precisión media: ",r6)

end
```

```
Promedio en los resultados utilizando precisión doble: 3.316899912297704
e-14
Promedio en los resultados utilizando precisión simple: 3.3111314e-7
Promedio en los resultados utilizando precisión media: 0.0003424
```

Resultados

En los experimentos se logra evidenciar que el error en el uso de cada uno de los formatos se hace menor al utilizar formatos más precisos. A diferencia que en la parte 1, al utilizar precisión media los valores no desbordan la pila de memoria.

Conclusiones sobre el desempeño de los métodos y el impacto de la precisión numérica

A nivel general no parece haber demasiada diferencia entre los métodos utilizados, se ha evidenciado que utilizando matrices definidas positivas, el método de **Minimal Residual** puede llegar a converger más rápido, sin embargo, puede tener muchas fluctuaciones en el comportamiento; mientras que utilizando matrices invertibles pero no definidas positivas, el método de **Residual Norm Steepest Descent** claramente converge más rápidamente.

Por otro lado, comparando los tiempos de ejecución, utilizando matrices definidas positivas, los tres métodos parecen ejecutarse en tiempos parecidos; mientras que utilizando matrices invertibles pero no definidas positivas, el método de **Residual Norm Steepest Descent** claramente se ejecuta mucho más rápido que el método de **Minimal Residual**.

Finalmente, también se evidencia que mientras se utilicen formatos con mayor precisión, los resultados son mejores. Esto sobre todo se observa utilizando matrices definidas positivas, en donde la diferencia de errores era más amplia entre formatos.

Reflexión

Durante el desarrollo de esta tarea, me enfrenté al desafío de comprender las sutilezas teóricas y prácticas de los métodos iterativos de proyección unidimensional. Al principio, las diferencias

entre los algoritmos de Descenso más Empinado (Steepest Descent), Residuo Mínimo (Minimal Residual) y Descenso más Empinado con Norma del Residuo (Residual Norm Steepest Descent) no eran del todo evidentes. Sin embargo, la necesidad de diseñar un marco experimental robusto me obligó a profundizar en su funcionamiento, especialmente en cómo su convergencia se ve afectada por las propiedades de la matriz, como la definicón positiva.

En comparación con talleres anteriores, mi manejo de Julia ha mejorado notablemente. Me sentí con más confianza para estructurar un análisis detallado, explorando no solo la velocidad de convergencia y el tiempo de cómputo, sino también el impacto de la precisión numérica (Float16, Float32, y Float64) en la estabilidad y el resultado final de los algoritmos.

Declaración de IA y fuentes externas

Se utilizó inteligencia artificial -en particular **ChatGPT de OpenAI** y **Gemini de Google**- como herramienta de apoyo para:

- Entender los algoritmos usados.
- Generar código en Julia conforme a algoritmos académicos.
- Organizar comparaciones conceptuales y computacionales.
- Corregir problemas de implementación de código en Julia.

La IA fue utilizada de manera responsable como herramienta de apoyo técnico y pedagógico.

Recursos usados

- **ChatGPT**
- **Gemini**
- *Iterative Methods for Sparse Linear Systems*, Yousef Saad