Numerical Linear Algebra for Computational Sciences and Information Engineering
CITHN2006

Jacobi-Davidson Methods

by Nicolas Venkovic

Chair of Computational Mathematics
School of Computation, Information and Technology (CIT)
Technical University of Munich, Germany

1. *Implement a generalized Davidson method equivalent to an Arnoldi procedure with Rayleigh-Ritz approximation of the most dominant eigenpair. That is, use the eigen-residual of the approximate eigenvector as expansion vector. Test your implementation with the Kuu matrix from the SuiteSparse collection.*

[1]:
```
using LinearAlgebra, Printf, SparseArrays, LinearOperators, Random
using MatrixMarket: mmread
using IterativeSolvers: gmres!
using IncompleteLU: ilu
using Plots
using Plots.PlotMeasures
Random.seed!(1);
```

[2]:
```
function RR_Arnoldi(A,
                    q::Vector{Float64},
                    k::Int,
                    tol::Float64;
                    ortho=:CGS2)
  n, _ = size(A)
  Q = zeros(Float64, (n, k+1))
  W = zeros(Float64, (n, k+1))
  B = zeros(Float64, (k+1, k+1))
  y = zeros(Float64, n)
  r = zeros(Float64, n)
  t = zeros(Float64, n)
  res = zeros(Float64, k+1)
  rho = zeros(Float64, k+1)
  t .= q
  j = 0
```

```
  while true
    # Orthogonalize t against Q[1:j]
    if ortho == :MGS
      for i in 1:j
        h = Q[:, i]'t
        t .-= h .* Q[:, i]
      end
    elseif ortho == :CGS2
      t .-= Q[:, 1:j] * (Q[:, 1:j]'t)
      t .-= Q[:, 1:j] * (Q[:, 1:j]'t)
    end
    Q[:, j+1] = t ./ norm(t)
    j += 1
    W[:, j] = A * Q[:, j]
    # Update projected matrix B for RR procedure
    B[1:j, j] = Q[:, 1:j]'W[:, j]
    B[j, 1:j-1] = Q[:, j]'W[:, 1:j-1]
    # B[1:j, 1:j] = Q[:,1:j]'W[:,1:j] = Q[:, 1:j]'A*Q[:, 1:j]
    vals, vecs = eigen(B[1:j, 1:j], sortby=norm)
    rho[j] = vals[j]
    y .= Q[:, 1:j] * vecs[:, j]
    r .= W[:, 1:j] * vecs[:, j] - rho[j] .* y
    res[j] = norm(r) / norm(rho[j])
    if ((res[j] < tol) || (j > k))
      break
    end
    # Expansion vector set along eigenresidual <=> RR Arnoldi
    t .= r
  end
  return res[1:j], rho[1:j], y
end;
```

[3]:
```
A = mmread("matrices/Kuu.mtx")
n = A.n
q = rand(n);
k = 200;
tol = 1e-8;
```

[4]:
```
res_RR_Arnoldi, val_RR_Arnoldi, _ = RR_Arnoldi(A, q, k, tol);
```

2. *Implement the Davidson method with Rayleigh-Ritz approximation of the most dominant eigenpair. Test your implementation with the* ***Kuu*** *matrix from the SuiteSparse collection.*

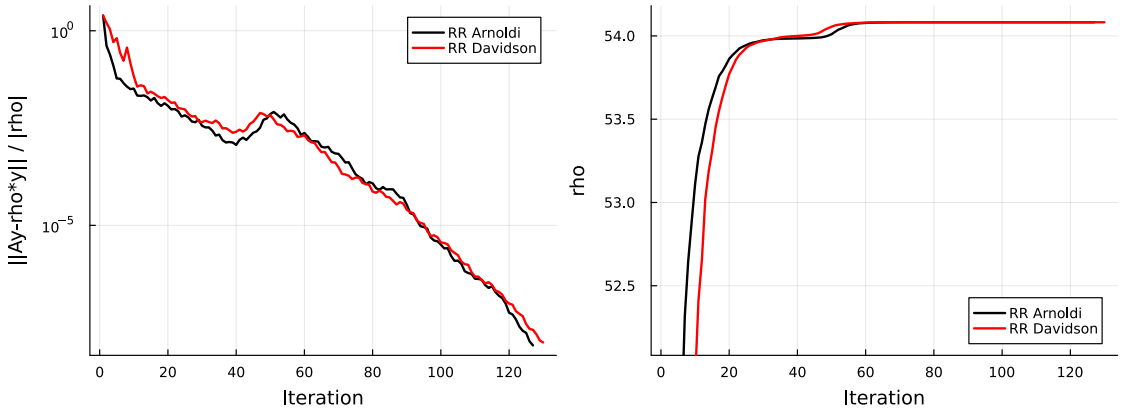
[5]:
```
function RR_Davidson(A,
                     q::Vector{Float64},
                     k::Int,
                     tol::Float64;
                     ortho=:CGS2)
```

```
  n, _ = size(A)
  Q = zeros(Float64, (n, k+1))
  W = zeros(Float64, (n, k+1))
  B = zeros(Float64, (k+1, k+1))
  y = zeros(Float64, n)
  r = zeros(Float64, n)
  t = zeros(Float64, n)
  res = zeros(Float64, k+1)
  rho = zeros(Float64, k+1)
  t .= q
  j = 0
  DA = spdiagm(diag(A, 0))
  while true
    # Orthogonalize t against Q[1:j]
    if ortho == :MGS
      for i in 1:j
        h = Q[:, i]'t
        t .-= h .* Q[:, i]
      end
    elseif ortho == :CGS2
      t .-= Q[:, 1:j] * (Q[:, 1:j]'t)
      t .-= Q[:, 1:j] * (Q[:, 1:j]'t)
    end
    Q[:, j+1] = t ./ norm(t)
    j += 1
    W[:, j] = A * Q[:, j]
    # Update projected matrix B for RR procedure
    B[1:j, j] = Q[:, 1:j]'W[:, j]
    B[j, 1:j-1] = Q[:, j]'W[:, 1:j-1]
    # B[1:j, 1:j] = Q[:,1:j]'W[:,1:j] = Q[:, 1:j]'A*Q[:, 1:j]
    vals, vecs = eigen(B[1:j, 1:j], sortby=norm)
    rho[j] = vals[j]
    y .= Q[:, 1:j] * vecs[:, j]
    r .= W[:, 1:j] * vecs[:, j] - rho[j] .* y
    res[j] = norm(r) / norm(rho[j])
    if ((res[j] < tol) || (j > k))
      break
    end
    # Compute expansion vector
    t .= (DA - rho[j] * I) \ r
  end
  return res[1:j], rho[1:j], y
end;
```

[6]:
```
res_RR_Davidson, val_RR_Davidson, _ = RR_Davidson(A, q, k, tol);
```

[7]:
```
p1 = plot([i for i in 1:length(res_RR_Arnoldi)], res_RR_Arnoldi, yscale=:log10,␣
 ↪xlabel="Iteration", ylabel="||Ay-rho*y|| / |rho|", label="RR Arnoldi",␣
 ↪linewidth=2, color=:black)
plot!([i for i in 1:length(res_RR_Davidson)], res_RR_Davidson, yscale=:log10,␣
 ↪label="RR Davidson", linewidth=2, color=:red)
p2 = plot([i for i in 1:length(val_RR_Arnoldi)], val_RR_Arnoldi,␣
 ↪xlabel="Iteration", ylabel="rho", label="RR Arnoldi", linewidth=2, color=:
 ↪black)
plot!([i for i in 1:length(val_RR_Davidson)], val_RR_Davidson, label="RR␣
 ↪Davidson", linewidth=2, color=:red, ylims=(val_RR_Davidson[end]-2,␣
 ↪val_RR_Davidson[end]+.1))
plot(p1, p2; layout=(1, 2), size=(950, 350), left_margin=3mm, bottom_margin=3mm)
```

[7]:



3. *Implement a generalized Davidson method with Rayleigh-Ritz approximation of the most dominant eigenpair. Use a preconditioned GMRES solver to compute the expansion vector. Test your implementation with the* `Kuu` *matrix from the SuiteSparse collection using different numbers of inner (GMRES) iterations. What do you observe?*

[8]:
```
function RR_GD(A,
               q::Vector{Float64},
               k::Int,
               tol::Float64;
               ortho=:CGS2,
               inner_precond=:Tridiagonal,
               inner_iters=5)
  n, _ = size(A)
  Q = zeros(Float64, (n, k+1))
  W = zeros(Float64, (n, k+1))
  B = zeros(Float64, (k+1, k+1))
  y = zeros(Float64, n)
  r = zeros(Float64, n)
  t = zeros(Float64, n)
  res = zeros(Float64, k+1)
```

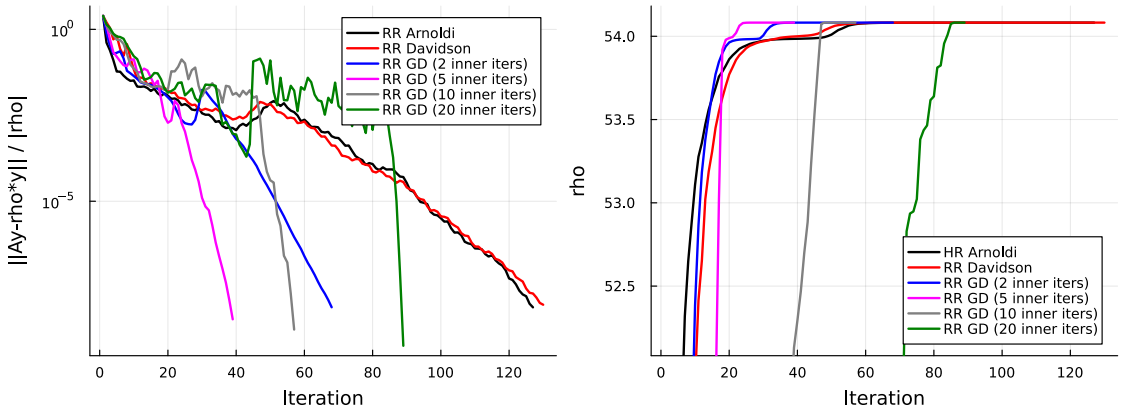
```
rho = zeros(Float64, k+1)
M = if inner_precond == :Diagonal
  Diagonal(A)
elseif inner_precond == :Tridiagonal
  Tridiagonal(A)
elseif inner_precond ==:ILU
  ilu(A)
end
t .= q
j = 0
while true
  # Orthogonalize t against Q[1:j]
  if ortho == :MGS
    for i in 1:j
      h = Q[:, i]'t
      t .-= h .* Q[:, i]
    end
  elseif ortho == :CGS2
    t .-= Q[:, 1:j] * (Q[:, 1:j]'t)
    t .-= Q[:, 1:j] * (Q[:, 1:j]'t)
  end
  Q[:, j+1] = t ./ norm(t)
  j += 1
  W[:, j] = A * Q[:, j]
  # Update projected matrix B for RR procedure
  B[1:j, j] = Q[:, 1:j]'W[:, j]
  B[j, 1:j-1] = Q[:, j]'W[:, 1:j-1]
  # B[1:j, 1:j] = Q[:,1:j]'W[:,1:j] = Q[:, 1:j]'A*Q[:, 1:j]
  vals, vecs = eigen(B[1:j, 1:j], sortby=norm)
  rho[j] = vals[j]
  y .= Q[:, 1:j] * vecs[:, j]
  r .= W[:, 1:j] * vecs[:, j] - rho[j] .* y
  r_norm = norm(r)
  res[j] = r_norm / norm(rho[j])
  if ((res[j] < tol) || (j > k))
    break
  end
  # Update preconditioner for correction equation
  if inner_precond == :Diagonal
    M .= Diagonal(A - rho[j] * I)
  elseif inner_precond == :Tridiagonal
    M .= Tridiagonal(A - rho[j] * I)
  end
  # Approximate solve of correction equation
  t .= r
  gmres!(t, A - rho[j] * I , -r, Pl=M, maxiter=inner_iters)
end
```

```
    return res[1:j], rho[1:j], y
end;
```

[9]:
```
res_RR_GD_2, val_RR_GD_2, _ = RR_GD(A, q, k, tol, inner_iters=2);
res_RR_GD_5, val_RR_GD_5, _ = RR_GD(A, q, k, tol, inner_iters=5);
res_RR_GD_10, val_RR_GD_10, _ = RR_GD(A, q, k, tol, inner_iters=10);
res_RR_GD_20, val_RR_GD_20, _ = RR_GD(A, q, k, tol, inner_iters=20);
```

[10]:
```
p1 = plot([i for i in 1:length(res_RR_Arnoldi)], res_RR_Arnoldi, yscale=:log10,␣
 ↪xlabel="Iteration", ylabel="||Ay-rho*y|| / |rho|", label="RR Arnoldi",␣
 ↪linewidth=2, color=:black)
plot!([i for i in 1:length(res_RR_Davidson)], res_RR_Davidson, yscale=:log10,␣
 ↪label="RR Davidson", linewidth=2, color=:red)
plot!([i for i in 1:length(res_RR_GD_2)], res_RR_GD_2, yscale=:log10, label="RR␣
 ↪GD (2 inner iters)", linewidth=2, color=:blue)
plot!([i for i in 1:length(res_RR_GD_5)], res_RR_GD_5, yscale=:log10, label="RR␣
 ↪GD (5 inner iters)", linewidth=2, color=:magenta)
plot!([i for i in 1:length(res_RR_GD_10)], res_RR_GD_10, yscale=:log10,␣
 ↪label="RR GD (10 inner iters)", linewidth=2, color=:gray)
plot!([i for i in 1:length(res_RR_GD_20)], res_RR_GD_20, yscale=:log10,␣
 ↪label="RR GD (20 inner iters)", linewidth=2, color=:green)
p2 = plot([i for i in 1:length(val_RR_Arnoldi)], val_RR_Arnoldi,␣
 ↪xlabel="Iteration", ylabel="rho", label="HR Arnoldi", linewidth=2, color=:
 ↪black)
plot!([i for i in 1:length(val_RR_Davidson)], val_RR_Davidson, label="RR␣
 ↪Davidson", linewidth=2, color=:red)
plot!([i for i in 1:length(val_RR_GD_2)], val_RR_GD_2, label="RR GD (2 inner␣
 ↪iters)", linewidth=2, color=:blue)
plot!([i for i in 1:length(val_RR_GD_5)], val_RR_GD_5, label="RR GD (5 inner␣
 ↪iters)", linewidth=2, color=:magenta)
plot!([i for i in 1:length(val_RR_GD_10)], val_RR_GD_10, label="RR GD (10 inner␣
 ↪iters)", linewidth=2, color=:gray)
plot!([i for i in 1:length(val_RR_GD_20)], val_RR_GD_20, label="RR GD (20 inner␣
 ↪iters)", linewidth=2, color=:green,␣
 ↪ylims=(val_RR_GD_20[end]-2,val_RR_GD_20[end]+.1))
plot(p1, p2; layout=(1, 2), size=(950, 350), left_margin=3mm, bottom_margin=3mm)
```

[10]:



Forming the expansion vector t by approximately solving the correction equation $(A - \rho I_n)t = -r$ with 5 GMRES iterations accelerates the convergence. But solving it too precisely, e.g., with 20 GMRES iterations leads to delay, or even stagnation.

4. *Implement a preconditioned Jacobi-Davidson method with Rayleigh-Ritz approximation of the most dominant eigenpair. Use a preconditioned GMRES solver to compute the expansion vector. Test your implementation with the* `Kuu` *matrix from the SuiteSparse collection using different numbers of (GMRES) inner iterations. What do you observe?*

[11]:
```
function mul_tA(A, lbda, y, x, tAx)
  # tAx .= (I - y*y')*(A - lbda*I)*(I - y*y')*x
  tAx .= x
  c = y'x
  tAx .-= c * y
  tAx .= (A - lbda * I) * tAx
  c = y'tAx
  tAx .-= c * y
end;

mutable struct OrthoPrecond
  T
  y::Vector{Float64}
end;

function LinearAlgebra.ldiv!(M::OrthoPrecond, z)
  z .= M.T \ z
  z .-= (M.y'z) * z
end;

function LinearAlgebra.ldiv!(Z, M::OrthoPrecond, R)
  ldiv!(M, R)
  Z .= R
end;
```

[12]:
```
function RR_JD(A,
               q::Vector{Float64},
               k::Int,
               tol::Float64;
               ortho=:CGS2,
               inner_precond=:Tridiagonal,
               inner_iters=5)
  n, _ = size(A)
  Q = zeros(Float64, (n, k+1))
  W = zeros(Float64, (n, k+1))
  B = zeros(Float64, (k+1, k+1))
  y = zeros(Float64, n)
  r = zeros(Float64, n)
  t = zeros(Float64, n)
  res = zeros(Float64, k+1)
  rho = zeros(Float64, k+1)
  M = if inner_precond == :Diagonal
    OrthoPrecond(Diagonal(A), copy(q))
  elseif inner_precond == :Tridiagonal
    OrthoPrecond(Tridiagonal(A), copy(q))
  end
  t .= q
  j = 0
  while true
    # Orthogonalize t against Q[1:j]
    if ortho == :MGS
      for i in 1:j
        h = Q[:, i]'t
        t .-= h .* Q[:, i]
      end
    elseif ortho == :CGS2
      t .-= Q[:, 1:j] * (Q[:, 1:j]'t)
      t .-= Q[:, 1:j] * (Q[:, 1:j]'t)
    end
    Q[:, j+1] = t ./ norm(t)
    j += 1
    W[:, j] = A * Q[:, j]
    # Update projected matrix B for RR procedure
    B[1:j, j] = Q[:, 1:j]'W[:, j]
    B[j, 1:j-1] = Q[:, j]'W[:, 1:j-1]
    # B[1:j, 1:j] = Q[:,1:j]'W[:,1:j] = Q[:, 1:j]'A*Q[:, 1:j]
    vals, vecs = eigen(B[1:j, 1:j], sortby=norm)
    rho[j] = vals[j]
    y .= Q[:, 1:j] * vecs[:, j]
    r .= W[:, 1:j] * vecs[:, j] - rho[j] .* y
    res[j] = norm(r) / norm(rho[j])
    if ((res[j] < tol) || (j > k))
```

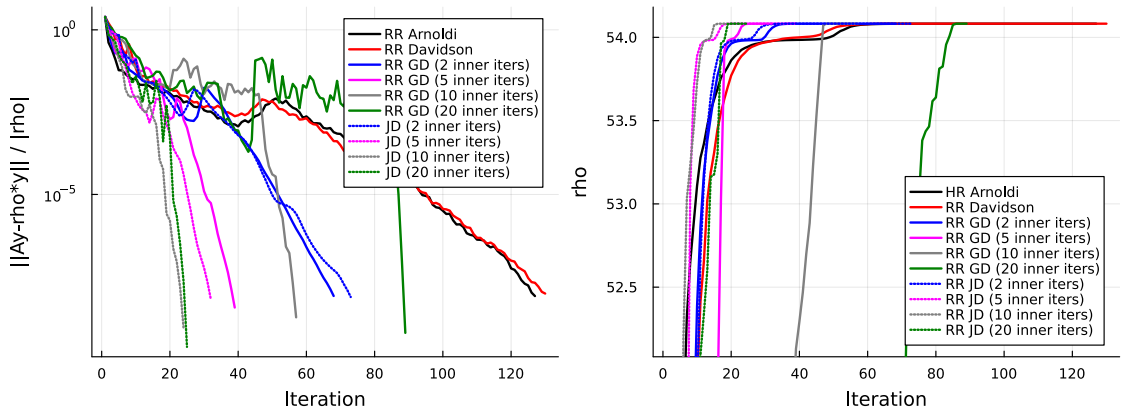
```
      break
    end
    # tA*x = (I - y*y')*(A - rho[j]*I)*(I - y*y')*x
    tA = LinearOperator(Float64, n, n, false, false,
                        (tAx, x) -> mul_tA(A, rho[j], y, x, tAx),
                        nothing, nothing)
    # Update preconditioner for correction equation
    if inner_precond == :Diagonal
      M.T .= Diagonal(A - rho[j] * I)
    elseif inner_precond == :Tridiagonal
      M.T .= Tridiagonal(A - rho[j] * I)
    end
    M.y .= y
    # Approximate solve of correction equation
    t .= r
    gmres!(t, tA, -r, Pl=M, maxiter=inner_iters)
  end
  return res[1:j], rho[1:j], y
end;
```

[13]:
```
res_RR_JD_2, val_RR_JD_2, _ = RR_JD(A, q, k, tol, inner_iters=2);
res_RR_JD_5, val_RR_JD_5, _ = RR_JD(A, q, k, tol, inner_iters=5);
res_RR_JD_10, val_RR_JD_10, _ = RR_JD(A, q, k, tol, inner_iters=10);
res_RR_JD_20, val_RR_JD_20, _ = RR_JD(A, q, k, tol, inner_iters=20);
```

[14]:
```
p1 = plot([i for i in 1:length(res_RR_Arnoldi)], res_RR_Arnoldi, yscale=:log10,␣
 ↪xlabel="Iteration", ylabel="||Ay-rho*y|| / |rho|", label="RR Arnoldi",␣
 ↪linewidth=2, color=:black)
plot!([i for i in 1:length(res_RR_Davidson)], res_RR_Davidson, yscale=:log10,␣
 ↪label="RR Davidson", linewidth=2, color=:red)
plot!([i for i in 1:length(res_RR_GD_2)], res_RR_GD_2, yscale=:log10, label="RR␣
 ↪GD (2 inner iters)", linewidth=2, color=:blue)
plot!([i for i in 1:length(res_RR_GD_5)], res_RR_GD_5, yscale=:log10, label="RR␣
 ↪GD (5 inner iters)", linewidth=2, color=:magenta)
plot!([i for i in 1:length(res_RR_GD_10)], res_RR_GD_10, yscale=:log10,␣
 ↪label="RR GD (10 inner iters)", linewidth=2, color=:gray)
plot!([i for i in 1:length(res_RR_GD_20)], res_RR_GD_20, yscale=:log10,␣
 ↪label="RR GD (20 inner iters)", linewidth=2, color=:green)
plot!([i for i in 1:length(res_RR_JD_2)], res_RR_JD_2, yscale=:log10, label="JD␣
 ↪(2 inner iters)", linewidth=2, color=:blue, linestyle=:dot)
plot!([i for i in 1:length(res_RR_JD_5)], res_RR_JD_5, yscale=:log10, label="JD␣
 ↪(5 inner iters)", linewidth=2, color=:magenta, linestyle=:dot)
plot!([i for i in 1:length(res_RR_JD_10)], res_RR_JD_10, yscale=:log10,␣
 ↪label="JD (10 inner iters)", linewidth=2, color=:gray, linestyle=:dot)
plot!([i for i in 1:length(res_RR_JD_20)], res_RR_JD_20, yscale=:log10,␣
 ↪label="JD (20 inner iters)", linewidth=2, color=:green, linestyle=:dot)
```

```
p2 = plot([i for i in 1:length(val_RR_Arnoldi)], val_RR_Arnoldi,␣
 ↪xlabel="Iteration", ylabel="rho", label="HR Arnoldi", linewidth=2, color=:
 ↪black)
plot!([i for i in 1:length(val_RR_Davidson)], val_RR_Davidson, label="RR␣
 ↪Davidson", linewidth=2, color=:red)
plot!([i for i in 1:length(val_RR_GD_2)], val_RR_GD_2, label="RR GD (2 inner␣
 ↪iters)", linewidth=2, color=:blue)
plot!([i for i in 1:length(val_RR_GD_5)], val_RR_GD_5, label="RR GD (5 inner␣
 ↪iters)", linewidth=2, color=:magenta)
plot!([i for i in 1:length(val_RR_GD_10)], val_RR_GD_10, label="RR GD (10 inner␣
 ↪iters)", linewidth=2, color=:gray)
plot!([i for i in 1:length(val_RR_GD_20)], val_RR_GD_20, label="RR GD (20 inner␣
 ↪iters)", linewidth=2, color=:green)
plot!([i for i in 1:length(val_RR_JD_2)], val_RR_JD_2, label="RR JD (2 inner␣
 ↪iters)", linewidth=2, color=:blue, linestyle=:dot)
plot!([i for i in 1:length(val_RR_JD_5)], val_RR_JD_5, label="RR JD (5 inner␣
 ↪iters)", linewidth=2, color=:magenta, linestyle=:dot)
plot!([i for i in 1:length(val_RR_JD_10)], val_RR_JD_10, label="RR JD (10 inner␣
 ↪iters)", linewidth=2, color=:gray, linestyle=:dot)
plot!([i for i in 1:length(val_RR_JD_20)], val_RR_JD_20, label="RR JD (20 inner␣
 ↪iters)", linewidth=2, color=:green, linestyle=:dot,␣
 ↪ylims=(val_RR_GD_20[end]-2,val_RR_GD_20[end]+.1))
plot(p1, p2; layout=(1, 2), size=(950, 350), left_margin=3mm, bottom_margin=3mm)
```

[14]:



[15]:

```
println("                          $(length(res_RR_Arnoldi)-1) Arnoldi ␣
 ↪iterations")
println("                          $(length(res_RR_Davidson)-1) Davidson␣
 ↪iterations")
println(" 2 inner iterations --- $(length(res_RR_GD_2)-1) GD outer iterations␣
 ↪=> $(length(res_RR_JD_2)-1) JD outer iterations")
```

```
println(" 5 inner iterations --- $(length(res_RR_GD_5)-1) GD outer iterations␣
  ↪=> $(length(res_RR_JD_5)-1) JD outer iterations")
println("10 inner iterations --- $(length(res_RR_GD_10)-1) GD outer iterations␣
  ↪=> $(length(res_RR_JD_10)-1) JD outer iterations")
println("20 inner iterations --- $(length(res_RR_GD_20)-1) GD outer iterations␣
  ↪=> $(length(res_RR_JD_20)-1) JD outer iterations")
```

```
                    126 Arnoldi  iterations
                    129 Davidson iterations
 2 inner iterations --- 67 GD outer iterations => 72 JD outer iterations
 5 inner iterations --- 38 GD outer iterations => 31 JD outer iterations
10 inner iterations --- 56 GD outer iterations => 23 JD outer iterations
20 inner iterations --- 88 GD outer iterations => 24 JD outer iterations
```

[16]:
```
println("Approximate eigenvalues")

println("                          Arnoldi: $(val_RR_Arnoldi[end])")
println("                         Davidson: $(val_RR_Arnoldi[end])")
println(" 2 inner iterations ---        GD: $(val_RR_GD_2[end])")
println("                                 JD: $(val_RR_JD_2[end])")
println(" 5 inner iterations ---        GD: $(val_RR_GD_5[end])")
println("                                 JD: $(val_RR_JD_5[end])")
println("10 inner iterations ---        GD: $(val_RR_GD_10[end])")
println("                                 JD: $(val_RR_JD_10[end])")
println("20 inner iterations ---        GD: $(val_RR_GD_20[end])")
println("                                 JD: $(val_RR_JD_20[end])")
```

```
Approximate eigenvalues
                          Arnoldi: 54.082055996412564
                         Davidson: 54.082055996412564
 2 inner iterations ---        GD: 54.08205599641252
                                  JD: 54.08205599641244
 5 inner iterations ---        GD: 54.08205599641254
                                  JD: 54.08205599641246
10 inner iterations ---        GD: 54.08205599641259
                                  JD: 54.08205599641256
20 inner iterations ---        GD: 54.08205599641251
                                  JD: 54.08205599641259
```

With proper projection through Jacobi-Davidson (JD) iteration, the convergence delay of the generalized Davidson (GD) iteration is partly remedied.

5. *Implement a Arnoldi procedure with harmonic Rtiz approximation of the eigen-pair closest to a shift σ. Test your implementation with the* `Kuu` *matrix from the SuiteSparse collection and $\sigma = 2$.*

[17]:
```
function HR_CGS2_arnoldi(A, q1::Vector{Float64}, k::Int, sigma, tol::Float64)
  n, _ = size(A)
  Q = zeros(Float64, (n, k+1))
  H = zeros(Float64, (k+1, k))
```

```
  h = zeros(Float64, k+1)
  w = zeros(Float64, n)
  res = zeros(Float64, k+1)
  val = zeros(Float64, k+1)
  Q[:, 1] = q1 ./ norm(q1)
  j = 0
  vec = nothing
  while true
    j += 1
    w .= A * Q[:, j]
    h .= Q'w
    h[j+1:k+1] .= 0.
    H[:, j] = h
    w -= Q * h
    h .= Q'w
    h[j+1:k+1] .= 0.
    w -= Q * h
    H[j + 1, j] = norm(w)
    Q[:, j + 1] = w ./ H[j + 1, j]
    e_j = zeros(Float64, j); e_j[j] = 1.
    f = (H[1:j, 1:j] - sigma * I)' \ e_j
    G = H[1:j, 1:j] + norm(H[j + 1, j])^2 .* f * e_j'
    vals, vecs = eigen(G)
    ind = argmin(norm.(vals .- sigma))
    val[j] = vals[ind]
    vec = vecs[:, ind]
    rho = val[j] - norm(H[j + 1, j])^2 * vec'f * vec[j]
    res[j] = sqrt.(real.((val[j] .- rho) .* conj.(rho .- sigma)))
    res[j] /= norm(rho)
    if ((res[j] < tol) || (j == k))
      break
    end
  end
  return res[1:j], val[1:j], Q[:, 1:j] * vec
end;
```

[18]:
```
sigma = 2.;
k = 500;
res_HR_Arnoldi, val_HR_Arnoldi, _ = HR_CGS2_arnoldi(A, q, k, sigma, tol);
```

6. *Implement the Davidson method with harmonic Ritz approximation of the eigenpair closest to . Test your implementation with the* `Kuu` *matrix from the SuiteSparse collection and a shift* $\sigma = 2$.

[19]:
```
function HR_Davidson(A,
                     q::Vector{Float64},
                     k::Int,
                     sigma::Float64,
                     tol::Float64;
```

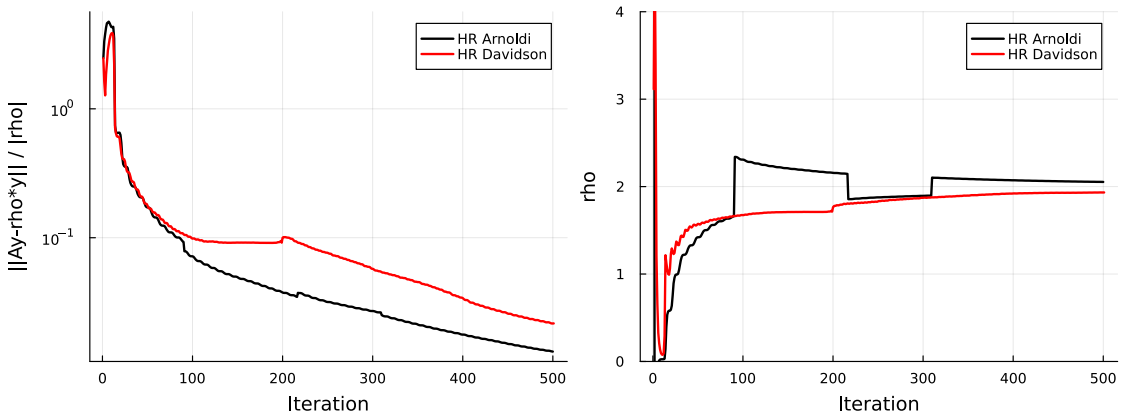
```
                    which=:RayleighQuotient)
n, _ = size(A)
Q = zeros(Float64, (n, k+1))
W = zeros(Float64, (n, k+1))
H = zeros(Float64, (k+1, k+1))
y = zeros(Float64, n)
r = zeros(Float64, n)
t = zeros(Float64, n)
res = zeros(Float64, k+1)
val = zeros(Float64, k+1)
t .= q
j = 0
DA = spdiagm(diag(A, 0))
while true
  j += 1
  w = (A - sigma * I) * t
  for i in 1:j
    witw = W[:, i]'w
    w .-= witw * W[:, i]
    t .-= witw * Q[:, i]
  end
  w_norm = norm(w)
  W[:, j] = w / w_norm
  Q[:, j] = t / w_norm
  H[j, 1:j] = W[:, j]'Q[:, 1:j]
  H[1:j, j] = W[:, 1:j]'Q[:, j]
  vals, vecs = eigen(H[1:j, 1:j])
  ind = argmax(norm.(vals))
  y .= Q[:, 1:j] * vecs[:, ind]
  yty = y'y
  y ./= sqrt(yty)
  if which == :HarmonicRitzValue
    dlbda = 1. / vals[ind]
    val[j] = sigma + dlbda
    r .= W[:, 1:j] * vecs[:, ind] / sqrt(yty) - dlbda * y
    # r .= A * y - val[j] * y
  elseif which == :RayleighQuotient
    drho = conj(vals[ind]) / yty
    val[j] = sigma + drho
    # val[j] = (A * y)'y
    r .= W[:, 1:j] * vecs[:, ind] / sqrt(yty) - drho * y
    # r .= A * y - val[j] * y
  end
  res[j] = norm(r) / norm(val[j])
  if ((res[j] < tol) || (j > k))
    break
  end
```

```
    t .= (DA - val[j] * I) \ r
  end
  return res[1:j], val[1:j], y
end;
```

[20]:
```
res_HR_Davidson, val_HR_Davidson, _ = HR_Davidson(A, q, k, sigma, tol, which=:
↪RayleighQuotient);
```

[21]:
```
p1 = plot([i for i in 1:length(res_HR_Arnoldi)], res_HR_Arnoldi, yscale=:log10,␣
↪xlabel="Iteration", ylabel="||Ay-rho*y|| / |rho|", label="HR Arnoldi",␣
↪linewidth=2, color=:black)
plot!([i for i in 1:length(res_HR_Davidson)], res_HR_Davidson, yscale=:log10,␣
↪label="HR Davidson", linewidth=2, color=:red)
p2 = plot([i for i in 1:length(val_HR_Arnoldi)], val_HR_Arnoldi,␣
↪xlabel="Iteration", ylabel="rho", label="HR Arnoldi", linewidth=2, color=:
↪black)
plot!([i for i in 1:length(val_HR_Davidson)], val_HR_Davidson, label="HR␣
↪Davidson", linewidth=2, color=:red, ylims=(sigma-2, sigma+2))
plot(p1, p2; layout=(1, 2), size=(950, 350), left_margin=3mm, bottom_margin=3mm)
```

[21]:



7. *Implement a generalized Davidson method with harmonic Ritz approximation of the eigenpair closest to a given shift σ. Use a preconditioned GMRES solver to compute the expansion vector. Test your implementation with the* `Kuu` *matrix from the SuiteSparse collection using different numbers of inner (GMRES) iterations and $\sigma = 2$.*

[22]:
```
function HR_GD(A,
               q::Vector{Float64},
               k::Int,
               sigma::Float64,
               tol::Float64;
               inner_precond=:Tridiagonal,
               inner_iters=5,
               which=:RayleighQuotient)
```

```
n, _ = size(A)
Q = zeros(Float64, (n, k+1))
W = zeros(Float64, (n, k+1))
H = zeros(Float64, (k+1, k+1))
y = zeros(Float64, n)
r = zeros(Float64, n)
t = zeros(Float64, n)
res = zeros(Float64, k+1)
val = zeros(Float64, k+1)
M = if inner_precond == :Diagonal
  Diagonal(A - sigma * I)
elseif inner_precond == :Tridiagonal
  Tridiagonal(A - sigma * I)
elseif inner_precond == :ILU
  ilu(A - sigma * I, τ=0.01)
end
t .= q
j = 0
while true
  j += 1
  w = (A - sigma * I) * t
  for i in 1:j
    witw = W[:, i]'w
    w .-= witw * W[:, i]
    t .-= witw * Q[:, i]
  end
  w_norm = norm(w)
  W[:, j] = w / w_norm
  Q[:, j] = t / w_norm
  # Update projected matrix H for HR procedure
  H[j, 1:j] = W[:, j]'Q[:, 1:j]
  H[1:j, j] = W[:, 1:j]'Q[:, j]
  vals, vecs = eigen(H[1:j, 1:j])
  ind = argmax(norm.(vals))
  y .= Q[:, 1:j] * vecs[:, ind]
  yty = y'y
  y ./= sqrt(yty)
  if which == :HarmonicRitzValue
    dlbda = 1. / vals[ind]
    val[j] = sigma + dlbda
    r .= W[:, 1:j] * vecs[:, ind] / sqrt(yty) - dlbda * y
    # r .= A * y - val[j] * y
  elseif which == :RayleighQuotient
    drho = conj(vals[ind]) / yty
    val[j] = sigma + drho
    # val[j] = (A * y)'y
    r .= W[:, 1:j] * vecs[:, ind] / sqrt(yty) - drho * y
```

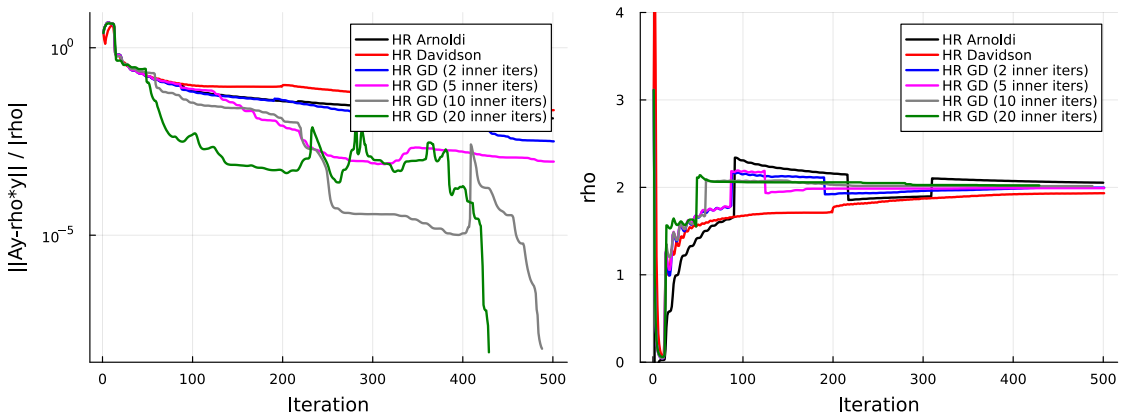
```
      # r .= A * y - val[j] * y
    end
    res[j] = norm(r) / norm(val[j])
    if ((res[j] < tol) || (j > k))
      break
    end
    # Compute expansion vector by GMRES
    t .= r
    gmres!(t, A - val[j] * I , -r, Pl=M, maxiter=inner_iters)
  end
  return res[1:j], val[1:j], y
end;
```

[23]:
```
res_HR_GD_2, val_HR_GD_2, _ = HR_GD(A, q, k, sigma, tol, inner_precond=:ILU,␣
 ↪inner_iters=2, which=:RayleighQuotient);
res_HR_GD_5, val_HR_GD_5, _ = HR_GD(A, q, k, sigma, tol, inner_precond=:ILU,␣
 ↪inner_iters=5, which=:RayleighQuotient);
res_HR_GD_10, val_HR_GD_10, _ = HR_GD(A, q, k, sigma, tol, inner_precond=:ILU,␣
 ↪inner_iters=10, which=:RayleighQuotient);
res_HR_GD_20, val_HR_GD_20, _ = HR_GD(A, q, k, sigma, tol, inner_precond=:ILU,␣
 ↪inner_iters=20, which=:RayleighQuotient);
```

[24]:
```
p1 = plot([i for i in 1:length(res_HR_Arnoldi)], res_HR_Arnoldi, yscale=:log10,␣
 ↪xlabel="Iteration", ylabel="||Ay-rho*y|| / |rho|", label="HR Arnoldi",␣
 ↪linewidth=2, color=:black)
plot!([i for i in 1:length(res_HR_Davidson)], res_HR_Davidson, yscale=:log10,␣
 ↪label="HR Davidson", linewidth=2, color=:red)
plot!([i for i in 1:length(res_HR_GD_2)], res_HR_GD_2, yscale=:log10, label="HR␣
 ↪GD (2 inner iters)", linewidth=2, color=:blue)
plot!([i for i in 1:length(res_HR_GD_5)], res_HR_GD_5, yscale=:log10, label="HR␣
 ↪GD (5 inner iters)", linewidth=2, color=:magenta)
plot!([i for i in 1:length(res_HR_GD_10)], res_HR_GD_10, yscale=:log10,␣
 ↪label="HR GD (10 inner iters)", linewidth=2, color=:gray)
plot!([i for i in 1:length(res_HR_GD_20)], res_HR_GD_20, yscale=:log10,␣
 ↪label="HR GD (20 inner iters)", linewidth=2, color=:green)
p2 = plot([i for i in 1:length(val_HR_Arnoldi)], val_HR_Arnoldi,␣
 ↪xlabel="Iteration", ylabel="rho", label="HR Arnoldi", linewidth=2, color=:
 ↪black)
plot!([i for i in 1:length(val_HR_Davidson)], val_HR_Davidson, label="HR␣
 ↪Davidson", linewidth=2, color=:red)
plot!([i for i in 1:length(val_HR_GD_2)], val_HR_GD_2, label="HR GD (2 inner␣
 ↪iters)", linewidth=2, color=:blue)
plot!([i for i in 1:length(val_HR_GD_5)], val_HR_GD_5, label="HR GD (5 inner␣
 ↪iters)", linewidth=2, color=:magenta)
plot!([i for i in 1:length(val_HR_GD_10)], val_HR_GD_10, label="HR GD (10 inner␣
 ↪iters)", linewidth=2, color=:gray)
```

```
plot!([i for i in 1:length(val_HR_GD_20)], val_HR_GD_20, label="HR GD (20 inner␣
 ↪iters)", linewidth=2, color=:green, ylims=(sigma-2, sigma+2))
plot(p1, p2; layout=(1, 2), size=(950, 350), left_margin=3mm, bottom_margin=3mm)
```

[24]:



8. *Implement a preconditioned Jacobi-Davidson method with harmonic Ritz approximation of the eigenpair closest to a given shift σ. Use a preconditioned GMRES solver to compute the expansion vector. Test your implementation with the* **Kuu** *matrix from the SuiteSparse collection using different numbers of inner (GMRES) iterations and $\sigma = 2$. What do you observe?*

[25]:

```
function HR_JD(A,
               q::Vector{Float64},
               k::Int,
               sigma::Float64,
               tol::Float64;
               inner_precond=:Tridiagonal,
               inner_iters=5,
               which=:RayleighQuotient)
  n, _ = size(A)
  Q = zeros(Float64, (n, k+1))
  W = zeros(Float64, (n, k+1))
  H = zeros(Float64, (k+1, k+1))
  y = zeros(Float64, n)
  r = zeros(Float64, n)
  t = zeros(Float64, n)
  res = zeros(Float64, k+1)
  val = zeros(Float64, k+1)
  M = if inner_precond == :Diagonal
    OrthoPrecond(Diagonal(A - sigma * I), copy(q))
  elseif inner_precond == :Tridiagonal
    OrthoPrecond(Tridiagonal(A - sigma * I), copy(q))
  elseif inner_precond == :ILU
    OrthoPrecond(ilu(A - sigma * I,  =0.01), copy(q))
  end
```

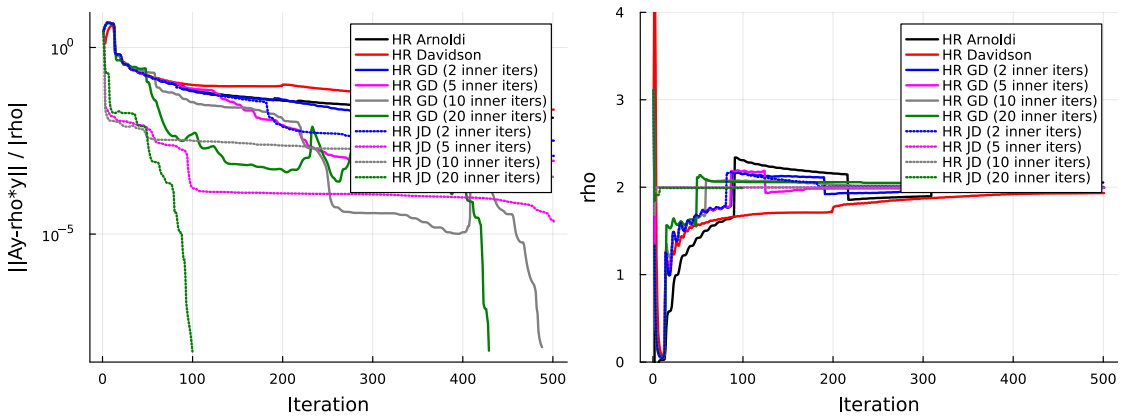
```
  t .= q
  j = 0
  while true
    j += 1
    w = (A - sigma * I) * t
    for i in 1:j
      witw = W[:, i]'w
      w .-= witw * W[:, i]
      t .-= witw * Q[:, i]
    end
    w_norm = norm(w)
    W[:, j] = w / w_norm
    Q[:, j] = t / w_norm
    H[j, 1:j] = W[:, j]'Q[:, 1:j]
    H[1:j, j] = W[:, 1:j]'Q[:, j]
    vals, vecs = eigen(H[1:j, 1:j])
    ind = argmax(norm.(vals))
    y .= Q[:, 1:j] * vecs[:, ind]
    yty = y'y
    y ./= sqrt(yty)
    if which == :HarmonicRitzValue
      dlbda = 1. / vals[ind]
      val[j] = sigma + dlbda
      r .= W[:, 1:j] * vecs[:, ind] / sqrt(yty) - dlbda * y
      # r .= A * y - val[j] * y
    elseif which == :RayleighQuotient
      drho = conj(vals[ind]) / yty
      val[j] = sigma + drho
      # val[j] = (A * y)'y
      r .= W[:, 1:j] * vecs[:, ind] / sqrt(yty) - drho * y
      # r .= A * y - val[j] * y
    end
    res[j] = norm(r) / norm(val[j])
    if ((res[j] < tol) || (j > k))
      break
    end
    tA = LinearOperator(Float64, n, n, false, false,
                        (tAx, x) -> mul_tA(A, val[j], y, x, tAx),
                        nothing, nothing)
    # Compute expansion vector by GMRES constrained
    # to the orthogonal complement of range(y)
    t .= r
    M.y .= y
    gmres!(t, tA, -r, Pl=M, maxiter=inner_iters)
  end
  return res[1:j], val[1:j], y
end;
```

[26]:
```
res_HR_JD_2, val_HR_JD_2, _ = HR_JD(A, q, k, sigma, tol, inner_precond=:ILU,␣
 ↪inner_iters=2, which=:RayleighQuotient);
res_HR_JD_5, val_HR_JD_5, _ = HR_JD(A, q, k, sigma, tol, inner_precond=:ILU,␣
 ↪inner_iters=5, which=:RayleighQuotient);
res_HR_JD_10, val_HR_JD_10, _ = HR_JD(A, q, k, sigma, tol, inner_precond=:ILU,␣
 ↪inner_iters=10, which=:RayleighQuotient);
res_HR_JD_20, val_HR_JD_20, _ = HR_JD(A, q, k, sigma, tol, inner_precond=:ILU,␣
 ↪inner_iters=20, which=:RayleighQuotient);
```

[27]:
```
p1 = plot([i for i in 1:length(res_HR_Arnoldi)], res_HR_Arnoldi, yscale=:log10,␣
 ↪xlabel="Iteration", ylabel="||Ay-rho*y|| / |rho|", label="HR Arnoldi",␣
 ↪linewidth=2, color=:black)
plot!([i for i in 1:length(res_HR_Davidson)], res_HR_Davidson, yscale=:log10,␣
 ↪label="HR Davidson", linewidth=2, color=:red)
plot!([i for i in 1:length(res_HR_GD_2)], res_HR_GD_2, yscale=:log10, label="HR␣
 ↪GD (2 inner iters)", linewidth=2, color=:blue)
plot!([i for i in 1:length(res_HR_GD_5)], res_HR_GD_5, yscale=:log10, label="HR␣
 ↪GD (5 inner iters)", linewidth=2, color=:magenta)
plot!([i for i in 1:length(res_HR_GD_10)], res_HR_GD_10, yscale=:log10,␣
 ↪label="HR GD (10 inner iters)", linewidth=2, color=:gray)
plot!([i for i in 1:length(res_HR_GD_20)], res_HR_GD_20, yscale=:log10,␣
 ↪label="HR GD (20 inner iters)", linewidth=2, color=:green)
plot!([i for i in 1:length(res_HR_JD_2)], res_HR_JD_2, yscale=:log10, label="HR␣
 ↪JD (2 inner iters)", linewidth=2, color=:blue, linestyle=:dot)
plot!([i for i in 1:length(res_HR_JD_5)], res_HR_JD_5, yscale=:log10, label="HR␣
 ↪JD (5 inner iters)", linewidth=2, color=:magenta, linestyle=:dot)
plot!([i for i in 1:length(res_HR_JD_10)], res_HR_JD_10, yscale=:log10,␣
 ↪label="HR JD (10 inner iters)", linewidth=2, color=:gray, linestyle=:dot)
plot!([i for i in 1:length(res_HR_JD_20)], res_HR_JD_20, yscale=:log10,␣
 ↪label="HR JD (20 inner iters)", linewidth=2, color=:green, linestyle=:dot)
p2 = plot([i for i in 1:length(val_HR_Arnoldi)], val_HR_Arnoldi,␣
 ↪xlabel="Iteration", ylabel="rho", label="HR Arnoldi", linewidth=2, color=:
 ↪black)
plot!([i for i in 1:length(val_HR_Davidson)], val_HR_Davidson, label="HR␣
 ↪Davidson", linewidth=2, color=:red)
plot!([i for i in 1:length(val_HR_GD_2)], val_HR_GD_2, label="HR GD (2 inner␣
 ↪iters)", linewidth=2, color=:blue)
plot!([i for i in 1:length(val_HR_GD_5)], val_HR_GD_5, label="HR GD (5 inner␣
 ↪iters)", linewidth=2, color=:magenta)
plot!([i for i in 1:length(val_HR_GD_10)], val_HR_GD_10, label="HR GD (10 inner␣
 ↪iters)", linewidth=2, color=:gray)
plot!([i for i in 1:length(val_HR_GD_20)], val_HR_GD_20, label="HR GD (20 inner␣
 ↪iters)", linewidth=2, color=:green)
plot!([i for i in 1:length(val_HR_JD_2)], val_HR_JD_2, label="HR JD (2 inner␣
 ↪iters)", linewidth=2, color=:blue, linestyle=:dot)
```

```
plot!([i for i in 1:length(val_HR_JD_5)], val_HR_JD_5, label="HR JD (5 inner␣
 ↪iters)", linewidth=2, color=:magenta, linestyle=:dot)
plot!([i for i in 1:length(val_HR_JD_10)], val_HR_JD_10, label="HR JD (10 inner␣
 ↪iters)", linewidth=2, color=:gray, linestyle=:dot)
plot!([i for i in 1:length(val_HR_JD_20)], val_HR_JD_20, label="HR JD (20 inner␣
 ↪iters)", linewidth=2, color=:green, linestyle=:dot, ylims=(sigma-2, sigma+2))
plot(p1, p2; layout=(1, 2), size=(950, 350), left_margin=3mm, bottom_margin=3mm)
```

[27]:



[28]:

```
println("                          $(length(res_HR_Arnoldi)-1) Arnoldi ␣
 ↪iterations")
println("                          $(length(res_HR_Davidson)-1) Davidson␣
 ↪iterations")
println(" 2 inner iterations --- $(length(res_HR_GD_2)-1) GD outer iterations␣
 ↪=> $(length(res_HR_JD_2)-1) JD outer iterations")
println(" 5 inner iterations --- $(length(res_HR_GD_5)-1) GD outer iterations␣
 ↪=> $(length(res_HR_JD_5)-1) JD outer iterations")
println("10 inner iterations --- $(length(res_HR_GD_10)-1) GD outer iterations␣
 ↪=> $(length(res_HR_JD_10)-1) JD outer iterations")
println("20 inner iterations --- $(length(res_HR_GD_20)-1) GD outer iterations␣
 ↪=> $(length(res_HR_JD_20)-1) JD outer iterations")
```

```
                     499 Arnoldi  iterations
                     500 Davidson iterations
 2 inner iterations --- 500 GD outer iterations => 500 JD outer iterations
 5 inner iterations --- 500 GD outer iterations => 500 JD outer iterations
10 inner iterations --- 487 GD outer iterations => 500 JD outer iterations
20 inner iterations --- 428 GD outer iterations => 99 JD outer iterations
```

[29]:

```
println("Approximate eigenvalues")

println("                           Arnoldi: $(val_HR_Arnoldi[end])")
println("                          Davidson: $(val_HR_Arnoldi[end])")
```

```
println(" 2 inner iterations ---        GD: $(val_HR_GD_2[end])")
println("                                JD: $(val_HR_JD_2[end])")
println(" 5 inner iterations ---        GD: $(val_HR_GD_5[end])")
println("                                JD: $(val_HR_JD_5[end])")
println("10 inner iterations ---         GD: $(val_HR_GD_10[end])")
println("                                JD: $(val_HR_JD_10[end])")
println("20 inner iterations ---         GD: $(val_HR_GD_20[end])")
println("                                JD: $(val_HR_JD_20[end])")
```

```
Approximate eigenvalues
                          Arnoldi: 2.053460565092543
                         Davidson: 2.053460565092543
 2 inner iterations ---         GD: 1.9967719080760646
                                JD: 2.004038551760178
 5 inner iterations ---         GD: 1.9907546977277473
                                JD: 1.9971231477084577
10 inner iterations ---         GD: 2.0144170225715863
                                JD: 1.996392400964148
20 inner iterations ---         GD: 2.0226994779640446
                                JD: 1.9908580446426472
```

For the case of 20 inner (GMRES) iterations, we see that the projection performed in the Jacobi-Davidson iteration leads to significant convergence acceleration compared to generalized Davidson iteration.