PracticeSession07

June 11, 2025

Numerical Linear Algebra for Computational Science and Information Engineering

Orthogonalization and Least-Squares Problems

Nicolas Venkovic (nicolas.venkovic@tum.edu)

[1]:
```
using LinearAlgebra, Plots, Printf, Latexify, LaTeXStrings, BenchmarkTools,␣
↪Random, SparseArrays
```

[2]:
```
function conditioned_matrix(m::Int, n::Int, cond::Float64)
  U, _ = qr(randn(m, m))
  V, _ = qr(randn(n, n))
  s = range(1.0, 1/cond; length=n)
  Σ = Diagonal(s)
  A = U[:, 1:n] * Σ * V'
  return A
end;
```

Exercise #1: Implement and test your implementation of HouseholderQR

[3]:
```
function HouseholderQR(A)
  m, n = size(A)
  Q = Matrix{Float64}(I, m, m)
  R = copy(A)
  for k = 1:n
    # Compute Householder vector
    v = R[k:m, k]
    v[1] -= sign(v[1]) * norm(v) # usage of sign(v[1]) allows to prevent␣
↪catastrophic cancellation
    beta = 2. / v'v
    # Apply Householder reflection
    R[k:m, k:n] -= beta .* v * (v'R[k:m, k:n])
    Q[1:m, k:m] -= Q[1:m, k:m] * v * (beta .* v')
  end
  return Q, R
end;
```

[4]:
```
m = 1_000;
n = 200;
Random.seed!(3);
A1 = conditioned_matrix(m, n, 1e8);
A2 = conditioned_matrix(m, n, 1e10);
```

[5]:
```
Q, R = @btime HouseholderQR(A1);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A1 - Q * R) norm(I - Q'Q)
```

```
  2.437 s (7807 allocations: 5.94 GiB)
||A - QR|| = 1.15E-14, ||I - Q'Q|| = 4.62E-14
```

[6]:
```
Q, R = @btime HouseholderQR(A2);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A2 - Q * R) norm(I - Q'Q)
```

```
  2.519 s (7807 allocations: 5.94 GiB)
||A - QR|| = 1.09E-14, ||I - Q'Q|| = 4.54E-14
```

Exercise #2: Implement CholeskyQR, CholeskyQR2 and Shifted CholeskyQR3

[7]:
```
function CholeskyQR(A)
  X = A'A
  chol = cholesky(X)
  Q = A / chol.U
  return Q, chol.U
end;
```

[8]:
```
function CholeskyQR2(A)
  Q1, R1 = CholeskyQR(A)
  Q, R2 = CholeskyQR(Q1)
  return Q, R2 * R1
end;
```

[9]:
```
function ShiftedCholeskyQR(A)
  X = A'A
  s = 11 * (m * n + n * (n + 1)) * eps() * norm(A)
  X += s * I
  chol = cholesky(X)
  Q = A / chol.U
  return Q, chol.U
end;
```

[10]:
```
function ShiftedCholeskyQR3(A)
  Q1, R1 = ShiftedCholeskyQR(A)
  Q, R2 = CholeskyQR2(Q1)
  return Q, R2 * R1
end;
```

Exercise #3: Compare loss of orthogonality and residual of each CholeskyQR implementation

[11]:
```
Q, R = @btime CholeskyQR(A1);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A1 - Q * R) norm(I - Q'Q)
```

```
  2.357 ms (10 allocations: 2.14 MiB)
||A - QR|| = 9.30E-16, ||I - Q'Q|| = 2.77E-02
```

[12]:
```
Q, R = @btime CholeskyQR(A2);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A1 - Q * R) norm(I - Q'Q)
```

```
PosDefException: matrix is not positive definite; Factorization failed.

Stacktrace:
  [1] checkpositivedefinite
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
 ↪LinearAlgebra/src/factorization.jl:68 [inlined]
  [2] #cholesky!#163
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
 ↪LinearAlgebra/src/cholesky.jl:269 [inlined]
  [3] cholesky!
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
 ↪LinearAlgebra/src/cholesky.jl:267 [inlined]
  [4] #cholesky!#164
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
 ↪LinearAlgebra/src/cholesky.jl:301 [inlined]
  [5] cholesky! (repeats 2 times)
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
 ↪LinearAlgebra/src/cholesky.jl:295 [inlined]
  [6] _cholesky
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
 ↪LinearAlgebra/src/cholesky.jl:411 [inlined]
  [7] cholesky(A::Matrix{Float64}, ::NoPivot; check::Bool)
    @ LinearAlgebra ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/
 ↪stdlib/v1.11/LinearAlgebra/src/cholesky.jl:401
  [8] cholesky (repeats 2 times)
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
 ↪LinearAlgebra/src/cholesky.jl:401 [inlined]
  [9] CholeskyQR(A::Matrix{Float64})
    @ Main ./In[7]:3
 [10] var"##core#256"()
    @ Main ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:598
 [11] var"##sample#257"(::Tuple{}, __params::BenchmarkTools.Parameters)
    @ Main ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:607
 [12] _lineartrial(b::BenchmarkTools.Benchmark, p::BenchmarkTools.Parameters;␣
 ↪maxevals::Int64, kwargs::@Kwargs{})
    @ BenchmarkTools ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:18
```

```
 [13] _lineartrial(b::BenchmarkTools.Benchmark, p::BenchmarkTools.Parameters)
    @ BenchmarkTools ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:18
 [14] #invokelatest#2
    @ ./essentials.jl:1055 [inlined]
 [15] invokelatest
    @ ./essentials.jl:1052 [inlined]
 [16] #lineartrial#46
    @ ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:51 [inlined]
 [17] lineartrial
    @ ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:50 [inlined]
 [18] tune!(b::BenchmarkTools.Benchmark, p::BenchmarkTools.Parameters;␣
↪progressid::Nothing, nleaves::Float64, ndone::Float64, verbose::Bool, pad::
↪String, kwargs::@Kwargs{})
    @ BenchmarkTools ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:29
 [19] tune!
    @ ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:288 [inlined]
 [20] tune!(b::BenchmarkTools.Benchmark)
    @ BenchmarkTools ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:28
 [21] top-level scope
    @ ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:728
```

[13]:
```julia
Q, R = @btime CholeskyQR2(A1);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A1 - Q * R) norm(I - Q'Q)
```

```
  4.780 ms (22 allocations: 4.58 MiB)
||A - QR|| = 3.35E-15, ||I - Q'Q|| = 5.57E-15
```

[14]:
```julia
Q, R = @btime CholeskyQR2(A2);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A2 - Q * R) norm(I - Q'Q)
```

```
PosDefException: matrix is not positive definite; Factorization failed.

Stacktrace:
  [1] checkpositivedefinite
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
 ↪LinearAlgebra/src/factorization.jl:68 [inlined]
  [2] #cholesky!#163
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
 ↪LinearAlgebra/src/cholesky.jl:269 [inlined]
  [3] cholesky!
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
 ↪LinearAlgebra/src/cholesky.jl:267 [inlined]
  [4] #cholesky!#164
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
 ↪LinearAlgebra/src/cholesky.jl:301 [inlined]
  [5] cholesky! (repeats 2 times)
```

```
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
↪LinearAlgebra/src/cholesky.jl:295 [inlined]
  [6] _cholesky
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
↪LinearAlgebra/src/cholesky.jl:411 [inlined]
  [7] cholesky(A::Matrix{Float64}, ::NoPivot; check::Bool)
    @ LinearAlgebra ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/
↪stdlib/v1.11/LinearAlgebra/src/cholesky.jl:401
  [8] cholesky (repeats 2 times)
    @ ~/.julia/juliaup/julia-1.11.5+0.x64.linux.gnu/share/julia/stdlib/v1.11/
↪LinearAlgebra/src/cholesky.jl:401 [inlined]
  [9] CholeskyQR(A::Matrix{Float64})
    @ Main ./In[7]:3
 [10] CholeskyQR2(A::Matrix{Float64})
    @ Main ./In[8]:2
 [11] var"##core#270"()
    @ Main ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:598
 [12] var"##sample#271"(::Tuple{}, __params::BenchmarkTools.Parameters)
    @ Main ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:607
 [13] _lineartrial(b::BenchmarkTools.Benchmark, p::BenchmarkTools.Parameters;␣
↪maxevals::Int64, kwargs::@Kwargs{})
    @ BenchmarkTools ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:18
 [14] _lineartrial(b::BenchmarkTools.Benchmark, p::BenchmarkTools.Parameters)
    @ BenchmarkTools ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:18
 [15] #invokelatest#2
    @ ./essentials.jl:1055 [inlined]
 [16] invokelatest
    @ ./essentials.jl:1052 [inlined]
 [17] #lineartrial#46
    @ ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:51 [inlined]
 [18] lineartrial
    @ ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:50 [inlined]
 [19] tune!(b::BenchmarkTools.Benchmark, p::BenchmarkTools.Parameters;␣
↪progressid::Nothing, nleaves::Float64, ndone::Float64, verbose::Bool, pad::
↪String, kwargs::@Kwargs{})
    @ BenchmarkTools ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:29
 [20] tune!
    @ ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:288 [inlined]
 [21] tune!(b::BenchmarkTools.Benchmark)
    @ BenchmarkTools ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:28
 [22] top-level scope
    @ ~/.julia/packages/BenchmarkTools/1i1mY/src/execution.jl:728
```

[15]:
```julia
Q, R = @btime ShiftedCholeskyQR3(A1);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A1 - Q * R) norm(I - Q'Q)
```

```
  7.750 ms (51 allocations: 7.33 MiB)
```

```
||A - QR|| = 3.89E-15, ||I - Q'Q|| = 4.35E-15
```

[16]:
```julia
Q, R = @btime ShiftedCholeskyQR3(A2);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A2 - Q * R) norm(I - Q'Q)
```

```
  8.398 ms (51 allocations: 7.33 MiB)
||A - QR|| = 3.93E-15, ||I - Q'Q|| = 4.27E-15
```

Exercise #4: Implement CGS, MGS and CGS2

[17]:
```julia
function CGS_naive(X::Array{Float64,2})
  n, m = size(X)
  Q = Array{Float64,2}(undef, n, m)
  R = zeros(Float64, m, m)
  R[1, 1] = norm(X[:, 1])
  Q[:, 1] = X[:, 1] ./ R[1, 1]
  for i in 2:m
    Q[:, i] = X[:, i]
    R[1:i-1, i] .= Q[:, 1:i-1]'X[:, i] # creates temporary array => very slow
    Q[:, i] -= Q[:, 1:i-1] * R[1:i-1, i] # creates temporary array => very slow
    R[i, i] = norm(Q[:, i])
    Q[:, i] ./= R[i, i]
  end
  return Q, R
end;

function CGS(X::Array{Float64,2})
  n, m = size(X)
  Q = zeros(Float64, n, m)
  R = zeros(Float64, m, m)
  r = zeros(Float64, m)
  w = zeros(Float64, n)
  R[1, 1] = norm(X[:, 1])
  Q[:, 1] = X[:, 1] ./ R[1, 1]
  for i in 2:m
    w .= X[:, i]
    r .= (w'Q)'
    r[i:m] .= 0.
    R[:, i] = r
    w -= Q * r
    R[i, i] = norm(w)
    Q[:, i] = w ./ R[i, i]
  end
  return Q, R
end;
```

[18]:
```julia
function MGS_naive(X::Array{Float64,2})
  n, m = size(X)
```

```julia
  Q = Array{Float64,2}(undef, n, m)
  R = zeros(Float64, m, m)
  R[1, 1] = norm(X[:, 1])
  Q[:, 1] = X[:, 1] ./ R[1, 1]
  for i in 2:m
    Q[:, i] = X[:, i]
    for j in 1:i-1
      R[j, i] = Q[:, j]'Q[:, i]
      Q[:, i] -= R[j, i] .* Q[:, j]
    end
    R[i, i] = norm(Q[:, i])
    Q[:, i] ./= R[i, i]
  end
  return Q, R
end;

function MGS(X::Array{Float64,2})
  n, m = size(X)
  Q = Array{Float64,2}(undef, n, m)
  q = Vector{Float64}(undef, n)
  R = zeros(Float64, m, m)
  w = Vector{Float64}(undef, n)
  R[1, 1] = norm(X[:, 1])
  Q[:, 1] = X[:, 1] ./ R[1, 1]
  for i in 2:m
    w .= X[:, i]
    for j in 1:i-1
      q .= Q[:, j]
      R[j, i] = q'w
      w -= R[j, i] .* q
    end
    R[i, i] = norm(w)
    Q[:, i] = w ./ R[i, i]
  end
  return Q, R
end;
```

[19]:
```julia
function CGS2_naive(X::Array{Float64,2})
  n, m = size(X)
  Q = Array{Float64,2}(undef, n, m)
  R = zeros(Float64, m, m)
  R[1, 1] = norm(X[:, 1])
  Q[:, 1] = X[:, 1] ./ R[1, 1]
  for i in 2:m
    Q[:, i] = X[:, i]
    R[1:i-1, i] .= Q[:, 1:i-1]'X[:, i] # creates temporary array => very slow
    Q[:, i] -= Q[:, 1:i-1] * R[1:i-1, i] # creates temporary array => very slow
```

```julia
    r = Q[:, 1:i-1]'Q[:, i] # creates temporary array => very slow
    Q[:, i] -= Q[:, 1:i-1] * r # creates temporary array => very slow
    R[i, i] = norm(Q[:, i])
    Q[:, i] ./= R[i, i]
  end
  return Q, R
end;

function CGS2(X::Array{Float64,2})
  n, m = size(X)
  Q = zeros(Float64, n, m)
  R = zeros(Float64, m, m)
  r = zeros(Float64, m)
  w = zeros(Float64, n)
  R[1, 1] = norm(X[:, 1])
  Q[:, 1] = X[:, 1] ./ R[1, 1]
  for i in 2:m
    w .= X[:, i]
    r .= Q'w
    r[i:m] .= 0.
    R[:, i] = r
    w -= Q * r
    r .= Q'w
    r[i:m] .= 0.
    w -= Q * r
    R[i, i] = norm(w)
    Q[:, i] = w ./ R[i, i]
  end
  return Q, R
end;
```

Exercise #5: Compare runtime, loss of orthogonality and residual of CGS, MGS and CGS2

[20]:
```julia
Q, R = @btime CGS_naive(A1);
Q, R = @btime CGS(A1);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A1 - Q * R) norm(I - Q'Q)
```

```
  55.181 ms (5985 allocations: 316.59 MiB)
  14.985 ms (2806 allocations: 8.31 MiB)
||A - QR|| = 2.45E-15, ||I - Q'Q|| = 9.54E-08
```

[21]:
```julia
Q, R = @btime CGS(A2);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A2 - Q * R) norm(I - Q'Q)
```

```
  14.345 ms (2806 allocations: 8.31 MiB)
||A - QR|| = 2.47E-15, ||I - Q'Q|| = 1.43E-05
```

[22]:
```
Q, R = @btime MGS_naive(A1);
Q, R = @btime MGS(A1);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A1 - Q * R) norm(I - Q'Q)
```

```
  140.893 ms (359807 allocations: 925.59 MiB)
  82.462 ms (180315 allocations: 464.51 MiB)
||A - QR|| = 3.81E-15, ||I - Q'Q|| = 1.35E-08
```

[23]:
```
Q, R = @btime MGS(A2);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A2 - Q * R) norm(I - Q'Q)
```

```
  73.142 ms (180315 allocations: 464.51 MiB)
||A - QR|| = 3.85E-15, ||I - Q'Q|| = 1.47E-06
```

[24]:
```
Q, R = @btime CGS2_naive(A1);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A1 - Q * R) norm(I - Q'Q)
Q, R = @btime CGS2(A1);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A1 - Q * R) norm(I - Q'Q)
```

```
  110.824 ms (9965 allocations: 626.57 MiB)
||A - QR|| = 3.30E-15, ||I - Q'Q|| = 5.73E-15  27.846 ms (4398 allocations:
11.69 MiB)
||A - QR|| = 3.30E-15, ||I - Q'Q|| = 5.55E-15
```

[25]:
```
Q, R = @btime CGS2(A2);
@printf "||A - QR|| = %.2E, ||I - Q'Q|| = %.2E" norm(A2 - Q * R) norm(I - Q'Q)
```

```
  28.367 ms (4398 allocations: 11.69 MiB)
||A - QR|| = 3.33E-15, ||I - Q'Q|| = 5.36E-15
```

Exercise #6: Implement Golub-Kahan bidiagonalization and verify matrix transformation

[26]:
```
function bidiagonalization(A, r0, k)
  m, n = size(A)
  U = zeros(Float64, m, k + 1)
  V = zeros(Float64, n, k + 1)
  beta = zeros(Float64, k + 1)
  alpha = zeros(Float64, k + 1)
  U[:, 1] = r0
  beta[1] = norm(U[:,1])
  U[:, 1] = U[:, 1] ./ beta[1]
  V[:, 1] = A'U[:, 1]
  alpha[1] = norm(V[:, 1])
  V[:, 1] = V[:, 1] ./ alpha[1]
  for i = 1:k
    U[:, i + 1] = A * V[:, i] - alpha[i] * U[:, i]
    beta[i + 1] = norm(U[:, i + 1])
    U[:, i + 1] = U[:, i + 1] / beta[i + 1]
```

```julia
    V[:, i + 1] = A'U[:, i + 1] - beta[i + 1] * V[:, i]
    alpha[i + 1] = norm(V[:, i + 1])
    V[:, i + 1] = V[:, i + 1] / alpha[i + 1]
  end
  return U, V[:, 1:k], alpha[1:k], beta
end;
```

[27]:
```julia
Random.seed!(3)
m = 100_000; n = 200;
A = sprand(Float64, m, n, .4);
b = rand(m);
```

[28]:
```julia
k = 5;
U, V, alpha, beta = bidiagonalization(A, rand(m), k);
B = spdiagm(k + 1, k, 0=>alpha, -1=>beta[2:k+1]);
norm(U'A*V - B)
```

[28]:
```
6.415087381394998
```

[29]:
```julia
U'A*V
```

[29]:
```
6×5 Matrix{Float64}:
 773.996          8.41039e-11   1.57414e-7   0.000298512   0.589457
 457.727          4.65152       9.30947e-8   0.00017654    0.348604
   9.09368e-10   96.5391        4.41971      9.91893e-8    0.000195861
   1.70404e-6     4.54171e-9   96.5037       4.36742       1.04521e-7
   0.00322698     8.59641e-6    4.58986e-9  96.6373        4.1883
   6.37841        0.0169915     9.07075e-6   4.78334e-9   96.5428
```

[30]:
```julia
B
```

[30]:
```
6×5 SparseMatrixCSC{Float64, Int64} with 10 stored entries:
 773.996
 457.727    4.65152
          96.5391    4.41971
                   96.5037    4.36742
                            96.6373    4.1883
                                     96.5428
```

[31]:
```julia
norm(I - U'U)
```

[31]:
```
0.010031473914636673
```

[32]:
```julia
norm(I - V'V)
```

[32]:
```
0.0010770355772659224
```

Exercise #7: Implement and test LSQR

[33]:
```
function LSQR(A, b, x0, k, tol)
  A_F = sqrt(sum(A.nzval.^2))
  m, n = size(A)
  U = zeros(Float64, m, k + 1)
  V = zeros(Float64, n, k)
  B = spdiagm(k + 1, k, 0=>zeros(Float64, k), -1=>zeros(Float64, k))
  U[:, 1] = b - A * x0
  beta1 = norm(U[:,1])
  U[:, 1] = U[:, 1] ./ beta1
  V[:, 1] = A'U[:, 1]
  B[1, 1] = norm(V[:, 1])
  V[:, 1] = V[:, 1] / B[1, 1]
  beta1_x_e1 = zeros(Float64, k + 1); beta1_x_e1[1] = beta1
  i = 0
  err = 1.
  yi = 1.
  while (err > tol && i < k)
    i += 1
    U[:, i + 1] = A * V[:, i] - B[i, i] * U[:, i]
    B[i + 1, i] = norm(U[:, i + 1])
    U[:, i + 1] = U[:, i + 1] / B[i + 1, i]
    V[:, i + 1] = A'U[:, i + 1] - B[i + 1, i] * V[:, i]
    B[i + 1, i + 1] = norm(V[:, i + 1])
    V[:, i + 1] = V[:, i + 1] / B[i + 1, i + 1]
    yi = B[1:i+1, 1:i] \ beta1_x_e1[1:i+1]
    ti = beta1_x_e1[1:i+1] - B[1:i+1, 1:i] * yi  # ||ti|| =  ||ri||
    si = B[1:i+1, 1:i+1]'ti                      # ||si|| = ||A'ri||
    err = norm(si) / (A_F * norm(ti))
    @printf "i = %i, ||A'ri||/(||A||.||ri||) = %.2E\n" i err
  end
  return x0 + V[:, 1:i] * yi
end;
```

[34]:
```
x = LSQR(A, b, rand(n), 100, 1e-5);
norm(b - A * x)
```

```
i = 1, ||A'ri||/(||A||.||ri||) = 5.75E-02
i = 2, ||A'ri||/(||A||.||ri||) = 1.07E-02
i = 3, ||A'ri||/(||A||.||ri||) = 4.53E-04
i = 4, ||A'ri||/(||A||.||ri||) = 2.17E-05
i = 5, ||A'ri||/(||A||.||ri||) = 1.07E-06
```

[34]: 92.84273159426289

[35]:
```
x = A \ b;
norm(A * x - b)
```

[35]: 92.84273158157164