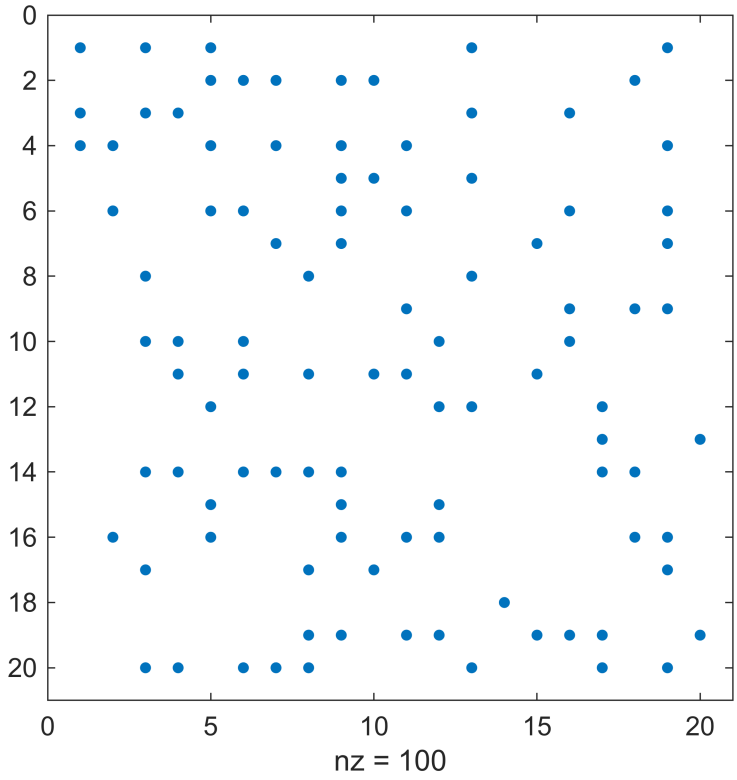
```
clear
rng('default')
n = 20;
density = 0.3;
A = sprand(n, n, density);
spy(A) % the "spy plot" exhibits the locations of the nonzero entries
```



MATLAB of course has built-in algorithms for calculating eigenvalues. These will be our benchmark to compare against.

```
lambda = sort(eig(full(A)), 'descend')
```

```
lambda = 20×1 complex
   2.4099 + 0.0000i
   1.0720 + 0.0000i
  -0.3626 + 0.9834i
  -0.3626 - 0.9834i
  -0.7316 + 0.6080i
  -0.7316 - 0.6080i
   0.3045 + 0.8830i
   0.3045 - 0.8830i
   0.8471 + 0.3529i
   0.8471 - 0.3529i
      ⋮
```

The Power Method

To kick off the power method we require an initial vector. A random vector is good a choice as any.

```
b = rand(n,1)
```

```
b = 20×1
    0.9516
    0.9203
    0.0527
    0.7379
    0.2691
    0.4228
    0.5479
    0.9427
    0.4177
    0.9831
      ⋮
```

We'll record the results of our iterations as columns of a matrix we'll call V. Better to normalise first.

```
beta = norm(b)
```

```
beta =
2.8645
```

```
V(:,1) = b / beta
```

```
V = 20×1
    0.3322
    0.3213
    0.0184
    0.2576
    0.0940
    0.1476
    0.1913
    0.3291
    0.1458
    0.3432
      ⋮
```

Now, time to iterate. We are going to continually multiply the vector we have by A. Let's generate a few more vectors this way, and record them as further columns of V.

```
m = 8;
for j = 1:m-1
    w = A * V(:,j);
    V(:,j+1) = w / norm(w);
end
V
```

```
V = 20×8
    0.3322    0.1834    0.1448    0.1573    0.1508    0.1520    0.1523    0.1528
    0.3213    0.2032    0.2376    0.2429    0.2587    0.2491    0.2556    0.2551
    0.0184    0.2129    0.1954    0.1927    0.1893    0.1963    0.1963    0.1957
    0.2576    0.4148    0.4079    0.3999    0.4120    0.4046    0.3996    0.4013
    0.0940    0.0881    0.0598    0.0708    0.0688    0.0704    0.0721    0.0725
    0.1476    0.2642    0.2425    0.2499    0.2490    0.2589    0.2592    0.2608
    0.1913    0.3275    0.2631    0.2677    0.2540    0.2516    0.2512    0.2517
    0.3291    0.1178    0.0611    0.0490    0.0437    0.0425    0.0428    0.0428
    0.1458    0.1529    0.1481    0.1510    0.1543    0.1600    0.1611    0.1601
    0.3432    0.2767    0.3954    0.3736    0.3576    0.3666    0.3678    0.3656
      ⋮
```

From the theory the iteration should converge to a multiple of the dominant eigenvector x_1 with eigenvalue λ_1. If so, the Rayleigh quotient $v^* A v$ of the final column above should be a good estimate of the associated eigenvalue $\mu \approx \lambda_1$.

```
v = V(:, end)
```

```
v = 20×1
    0.1528
    0.2551
    0.1957
    0.4013
    0.0725
    0.2608
    0.2517
    0.0428
    0.1601
    0.3656
      ⋮
```

```
mu = v'*A*v
```

```
mu =
2.4109
```

How does this compare to the true value λ_1?

```
lambda(1)
```

```
ans =
2.4099
```

Not too bad really.

How about the eigenvector? We can check the *eigenvalue residual:* how close is Av to μv?

```
power_method_residual = norm(A*v - mu*v)
```

```
power_method_residual =
0.0049
```

So, the computed eigenpair μ, v is reasonable. Further iterations of the power method would continue to refine these estimates, with the rate of convergence controlled by the ratio $|\lambda_2/\lambda_1|$.

Arnoldi process (Gram-Schmidt on the fly)

The Arnoldi process generates an orthonormal basis for the Krylov subspace, one vector at a time. First, compute the first column of V: it's just the scaled initial vector.

```
clear V
V(:,1) = b / beta
```

```
V = 20×1
    0.3322
    0.3213
    0.0184
    0.2576
```

```
    0.0940
    0.1476
    0.1913
    0.3291
    0.1458
    0.3432
     ⋮
```

As before, we generate the next vector by multiplying the previous vector by A.

```
w = A * V(:,1)
```

```
w = 20×1
    0.4393
    0.4865
    0.5098
    0.9934
    0.2109
    0.6327
    0.7843
    0.2821
    0.3661
    0.6627
     ⋮
```

Now though, before we include it in the matrix V, we orthonormalise it against the first vector. That is, we subtract away its projection onto v_1 and scale the result by its length.

$$w \rightarrow w - (w \cdot v_1)v_1$$

$$v_2 = w/||w||$$

We'll record the two intermediate scalars we needed in these calculations: $(w \cdot v_1)$ and $||w||$. Let's store them in an array H.

```
% orthogonalise
H(1,1) = dot(w, V(:,1))
```

```
H =
1.9646
```

```
w = w - H(1,1) * V(:,1)
```

```
w = 20×1
   -0.2134
   -0.1447
    0.4737
    0.4873
    0.0263
    0.3427
    0.4085
   -0.3645
    0.0796
   -0.0115
     ⋮
```

```
% normalise
```

```
H(2,1) = norm(w)
```

```
H = 2×1
    1.9646
    1.3690
```

```
V(:,2) = w / H(2,1)
```

```
V = 20×2
    0.3322   -0.1559
    0.3213   -0.1057
    0.0184    0.3460
    0.2576    0.3560
    0.0940    0.0192
    0.1476    0.2503
    0.1913    0.2984
    0.3291   -0.2662
    0.1458    0.0581
    0.3432   -0.0084
     ⋮
```

Automating this process to loop m times, we have derived the Arnoldi algorithm.

```
m = 8;
for j = 1:m

    % Compute next vector
    w = A * V(:,j);

    % Orthogonalise against previous columns (MGS)
    for i = 1:j
        H(i,j) = dot(w, V(:,i));
        w = w - H(i,j) * V(:,i);
    end

    % Normalise
    H(j+1, j) = norm(w);

    % Record the new vector in V
    V(:, j+1) = w / H(j+1, j);

end
V
```

```
V = 20×9
    0.3322   -0.1559   -0.2144    0.2110   -0.0406    0.0756   -0.0439   -0.0091 ···
    0.3213   -0.1057    0.1189    0.2512    0.3529   -0.2368    0.3470   -0.2012
    0.0184    0.3460    0.0062   -0.1915   -0.0701    0.2525    0.2622   -0.1374
    0.2576    0.3560    0.0499   -0.1874    0.2462   -0.2293   -0.4915    0.2289
    0.0940    0.0192   -0.1216    0.0975    0.0360    0.0853    0.1959    0.0495
    0.1476    0.2503   -0.0380   -0.0058    0.0384    0.4569    0.2185    0.3575
    0.1913    0.2984   -0.2149   -0.1397   -0.2171   -0.2496    0.1447    0.2912
    0.3291   -0.2662   -0.3199   -0.1088   -0.1071   -0.0081    0.0219   -0.0148
    0.1458    0.0581   -0.0101    0.0394    0.0956    0.3240    0.1619   -0.4883
    0.3432   -0.0084    0.5113    0.0477   -0.5563    0.1836    0.0564    0.1206
     ⋮
```

The columns of V form an orthonormal basis for the Krylov subspace.

We also have a record of all the dot products and norms we used in the process of computing V stored in the upper Hessenberg matrix H.

```
H
```

```
H = 9×8
    1.9646    0.8677   -0.6619    0.2143   -0.0086    0.5492    0.1972   -0.4080
    1.3690    0.3440    0.0313    0.1527    0.0970   -0.0530   -0.0858    0.0894
         0    1.0182   -0.3187    0.0068    0.2374    0.3432   -0.0341    0.2711
         0         0    0.8028   -0.0622   -0.1851    0.2660    0.0060    0.5394
         0         0         0    1.4113   -0.1300   -0.0222    0.3073   -0.1723
         0         0         0         0    1.1299    0.3552    0.0542   -0.4207
         0         0         0         0         0    1.0861   -0.0333   -0.0460
         0         0         0         0         0         0    0.7532   -0.4565
         0         0         0         0         0         0         0    0.8061
```

Notice that H is *rectangular*: it has $m+1$ rows and m columns. In the theory we denote it $\overline{H}_m$ to emphasise this.

Three Arnoldi relations

We can confirm the three Arnoldi relations.

1. $AV_m = V_{m+1}\overline{H}_m$

```
lhs = A * V(:, 1:m)
```

```
lhs = 20×8
    0.4393    0.0163    0.0130   -0.0245   -0.0173    0.1534    0.0737   -0.0690
    0.4865    0.3635   -0.0522    0.5360   -0.3446    0.5746    0.0024    0.4865
    0.5098    0.1413   -0.1570   -0.0302    0.3647    0.3190   -0.1475   -0.0290
    0.9934    0.3968   -0.3257    0.4691   -0.2123   -0.5308    0.2694    0.0220
    0.2109   -0.0357    0.0555    0.0669    0.0458    0.2770    0.0680   -0.3981
    0.6327    0.1755   -0.0824    0.1242    0.5263    0.4520    0.3074   -0.3518
    0.7843    0.0498   -0.1610   -0.2125   -0.2517    0.0516    0.1528   -0.0990
    0.2821   -0.1317   -0.2116   -0.1167   -0.0797    0.0795    0.0528   -0.0723
    0.3661    0.1362   -0.0598    0.1726    0.3483    0.3728   -0.3019    0.0881
    0.6627    0.8155   -0.3520   -0.7123    0.3885    0.5159   -0.0207    0.1022
     ⋮
```

```
rhs = V(:, 1:m+1) * H
```

```
rhs = 20×8
    0.4393    0.0163    0.0130   -0.0245   -0.0173    0.1534    0.0737   -0.0690
    0.4865    0.3635   -0.0522    0.5360   -0.3446    0.5746    0.0024    0.4865
    0.5098    0.1413   -0.1570   -0.0302    0.3647    0.3190   -0.1475   -0.0290
    0.9934    0.3968   -0.3257    0.4691   -0.2123   -0.5308    0.2694    0.0220
    0.2109   -0.0357    0.0555    0.0669    0.0458    0.2770    0.0680   -0.3981
    0.6327    0.1755   -0.0824    0.1242    0.5263    0.4520    0.3074   -0.3518
    0.7843    0.0498   -0.1610   -0.2125   -0.2517    0.0516    0.1528   -0.0990
    0.2821   -0.1317   -0.2116   -0.1167   -0.0797    0.0795    0.0528   -0.0723
    0.3661    0.1362   -0.0598    0.1726    0.3483    0.3728   -0.3019    0.0881
    0.6627    0.8155   -0.3520   -0.7123    0.3885    0.5159   -0.0207    0.1022
     ⋮
```

```
norm(lhs - rhs)
```

```
ans =
2.4116e-16
```

2. $AV_m = V_m H_m + h_{m+1,m}\, v_{m+1}\, e_m{}^T$

```
em = [zeros(m-1,1); 1];
lhs = A * V(:, 1:m)
```

```
lhs = 20×8
    0.4393    0.0163    0.0130   -0.0245   -0.0173    0.1534    0.0737   -0.0690
    0.4865    0.3635   -0.0522    0.5360   -0.3446    0.5746    0.0024    0.4865
    0.5098    0.1413   -0.1570   -0.0302    0.3647    0.3190   -0.1475   -0.0290
    0.9934    0.3968   -0.3257    0.4691   -0.2123   -0.5308    0.2694    0.0220
    0.2109   -0.0357    0.0555    0.0669    0.0458    0.2770    0.0680   -0.3981
    0.6327    0.1755   -0.0824    0.1242    0.5263    0.4520    0.3074   -0.3518
    0.7843    0.0498   -0.1610   -0.2125   -0.2517    0.0516    0.1528   -0.0990
    0.2821   -0.1317   -0.2116   -0.1167   -0.0797    0.0795    0.0528   -0.0723
    0.3661    0.1362   -0.0598    0.1726    0.3483    0.3728   -0.3019    0.0881
    0.6627    0.8155   -0.3520   -0.7123    0.3885    0.5159   -0.0207    0.1022
     ⋮
```

```
rhs = V(:, 1:m) * H(1:m, 1:m) + H(m+1,m) * V(:, m+1) * em'
```

```
rhs = 20×8
    0.4393    0.0163    0.0130   -0.0245   -0.0173    0.1534    0.0737   -0.0690
    0.4865    0.3635   -0.0522    0.5360   -0.3446    0.5746    0.0024    0.4865
    0.5098    0.1413   -0.1570   -0.0302    0.3647    0.3190   -0.1475   -0.0290
    0.9934    0.3968   -0.3257    0.4691   -0.2123   -0.5308    0.2694    0.0220
    0.2109   -0.0357    0.0555    0.0669    0.0458    0.2770    0.0680   -0.3981
    0.6327    0.1755   -0.0824    0.1242    0.5263    0.4520    0.3074   -0.3518
    0.7843    0.0498   -0.1610   -0.2125   -0.2517    0.0516    0.1528   -0.0990
    0.2821   -0.1317   -0.2116   -0.1167   -0.0797    0.0795    0.0528   -0.0723
    0.3661    0.1362   -0.0598    0.1726    0.3483    0.3728   -0.3019    0.0881
    0.6627    0.8155   -0.3520   -0.7123    0.3885    0.5159   -0.0207    0.1022
     ⋮
```

```
norm(lhs - rhs)
```

```
ans =
2.4045e-16
```

3. $V_m^* A V_m = H_m$

```
lhs = V(:, 1:m)' * A * V(:, 1:m)
```

```
lhs = 8×8
    1.9646    0.8677   -0.6619    0.2143   -0.0086    0.5492    0.1972   -0.4080
    1.3690    0.3440    0.0313    0.1527    0.0970   -0.0530   -0.0858    0.0894
    0.0000    1.0182   -0.3187    0.0068    0.2374    0.3432   -0.0341    0.2711
   -0.0000   -0.0000    0.8028   -0.0622   -0.1851    0.2660    0.0060    0.5394
   -0.0000   -0.0000    0.0000    1.4113   -0.1300   -0.0222    0.3073   -0.1723
   -0.0000   -0.0000    0.0000   -0.0000    1.1299    0.3552    0.0542   -0.4207
    0.0000    0.0000   -0.0000    0.0000    0.0000    1.0861   -0.0333   -0.0460
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.7532   -0.4565
```

```
rhs = H(1:m, 1:m)
```

```
rhs = 8×8
    1.9646    0.8677   -0.6619    0.2143   -0.0086    0.5492    0.1972   -0.4080
    1.3690    0.3440    0.0313    0.1527    0.0970   -0.0530   -0.0858    0.0894
         0    1.0182   -0.3187    0.0068    0.2374    0.3432   -0.0341    0.2711
         0         0    0.8028   -0.0622   -0.1851    0.2660    0.0060    0.5394
         0         0         0    1.4113   -0.1300   -0.0222    0.3073   -0.1723
         0         0         0         0    1.1299    0.3552    0.0542   -0.4207
         0         0         0         0         0    1.0861   -0.0333   -0.0460
```

```
         0         0         0         0         0         0    0.7532   -0.4565
```

```
norm(lhs - rhs)
```

```
ans =
6.3275e-16
```

Ritz values

We can find the eigenvalues of H_m (*Ritz values*) and see how they compare to the true eigenvalues of A we found earlier.

```
ritz = sort(eig(H(1:m, 1:m)), 'descend')
```

```
ritz = 8×1 complex
   2.4106 + 0.0000i
   1.0533 + 0.0000i
  -0.3608 + 0.8559i
  -0.3608 - 0.8559i
  -0.8693 + 0.0000i
  -0.4590 + 0.0000i
   0.1246 + 0.4187i
   0.1246 - 0.4187i
```

```
lambda(1:m) % these are the first m true eigenvalues of A
```

```
ans = 8×1 complex
   2.4099 + 0.0000i
   1.0720 + 0.0000i
  -0.3626 + 0.9834i
  -0.3626 - 0.9834i
  -0.7316 + 0.6080i
  -0.7316 - 0.6080i
   0.3045 + 0.8830i
   0.3045 - 0.8830i
```

The first few eigenvalues are actually a pretty good match.

Since H_m is small, we could compute its whole eigen decomposition. But to keep things simple here, we'll use `eigs` to just get the dominant eigenvector of H_m.

```
[y, ~] = eigs(H(1:m, 1:m), 1, 'largestabs')
```

```
y = 8×1
   -0.8081
   -0.5445
   -0.2094
   -0.0679
   -0.0385
   -0.0209
   -0.0092
   -0.0024
```

To form the eigenvector estimate (*Ritz vector)*, we need to find the vector in $\mathscr{K}_m$ represented by these coordinates:

```
u = V(:, 1:m) * y
```

```
u = 20×1
```

```
   -0.1526
   -0.2554
   -0.1962
   -0.4004
   -0.0727
   -0.2611
   -0.2510
   -0.0425
   -0.1608
   -0.3663
     ⋮
     ⋮
```

Now, remember the power method residual:

```
power_method_residual
```

```
power_method_residual =
0.0049
```

Let's compare this to the Arnoldi method residual:

```
arnoldi_method_residual = norm(A*u - ritz(1)*u)
```

```
arnoldi_method_residual =
0.0020
```

From the theory, this residual should be given by $|h_{m+1,m}||y_m^{(m)}|$

```
abs( H(m+1,m) * y(m) )
```

```
ans =
0.0020
```