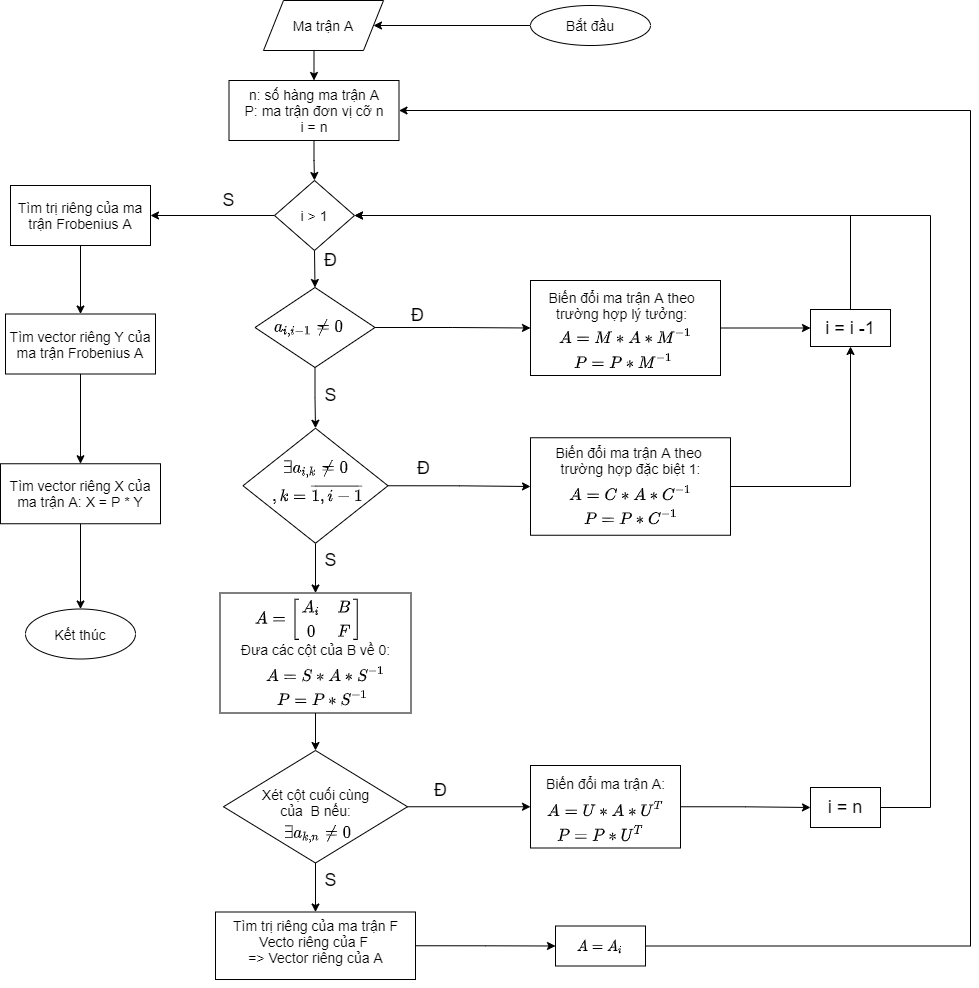
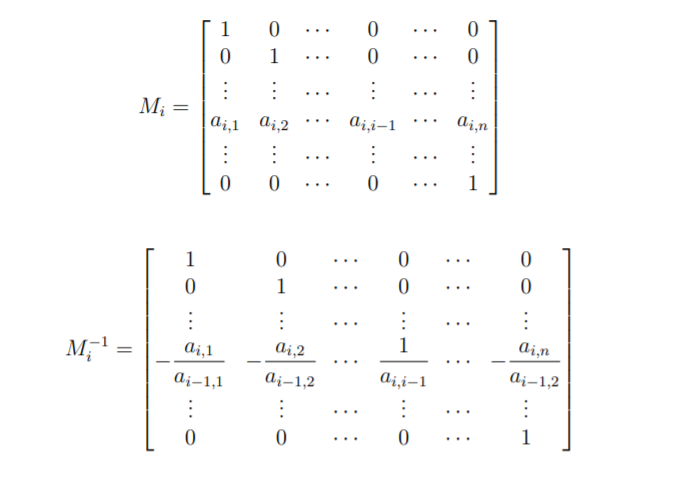
**Phương pháp Danilevski để tìm đúng giá trị riêng và vector riêng của ma trận vuông**

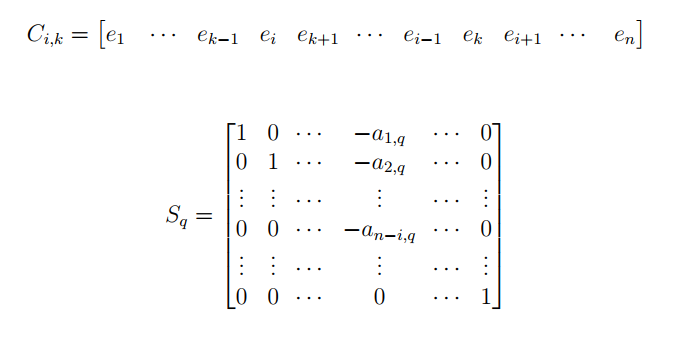
1. Thuật toán tổng thể

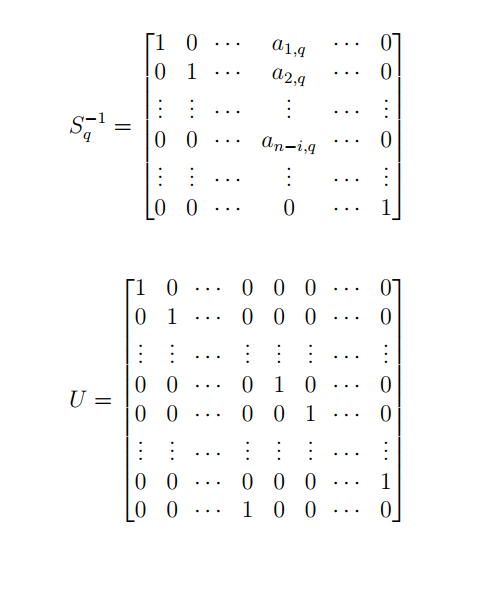


Ma trận M, C, S, U là các ma trận cụ thể trong phương pháp.

Tại lần lặp thứ i: C^-1 = C







1. Thuật toán chi tiết.

|  |
| --- |
| 1. Hàm tạo ma trận đơn vị:   Input: cỡ ma trận: n  Output: ma trận đơn vị cỡ: n  Function create\_init:   1. Hàm đổi chỗ 2 hàng của ma trận:   Input: ma trận A, hàng: x, y  Output: ma trận sau khi đổi chỗ  Function swap\_row:   1. Hàm đổi chỗ 2 cột của ma trận:   Input: ma trận A, cột: x, y  Output: ma trận sau khi đổi chỗ  Function swap\_col:   1. Hàm giải phương trình đa thức đặc trưng của ma trận Frobenius   Input: ma trận Frobenius A  Output: mảng chứa các trị riêng: eigenvalue[]  Function solution\_eigen:  #Mô tả các bước thực hiện của hàm.   * Ma trận A có kích cỡ n * Thực hiện nạp dữ liệu vào 1 mảng a[] có kích thước n + 1 * Mảng a[] chứa các hệ số của đa thức ma trận Frobenius A theo như công thức: a[i + 1] = (-1)^n \* (- A1,i), i = 1 -> n; a[1] = (-1)^n * Giải phương trình đa thức với hệ số là mảng a[] theo phương pháp: * B1: Tìm miền chứa nghiệm => thêm 2 đầu mút vào mảng b[] * B2: Tìm các cực trị của hàm số f(x) theo pp Gradient Descent => thêm các cực trị vào mảng b[] * Kiểm tra tính hợp lệ của khoảng cách li nghiệm là 2 giá trị b[i] và b[i+1]. Nếu b[i] và b[i+1] thỏa mãn tính chất kcl thì áp dụng thuật toán chia đôi để tìm nghiệm của pt f(x) = 0 * Thêm các nghiệm tìm được vào mảng eigenvalue[]  1. Hàm tìm trị riêng và vector riêng theo phương pháp Danilevski   Input: ma trận vuông A  Output: in ra trị riêng eigenvalue, vector riêng eigenvector  Function danilevski:  row = số hàng của ma trận A input  n = row  P = create\_init(n) //ma trận P là ma trận đơn vị  i = n  while i > 1:  if Ai,i-1  0:  M = create\_init(n)  M1 = create\_init(n)  for j = 1 to n:  Mi-1,j = Ai,j  M1i-1,j = - Ai,j / Ai,i-1  M1i-1,i-1 = 1/Ai,i-1  nhân 3 ma trận vuông: A = M\*A\*M1  nhân 2 ma trận vuông: P = P\*M1  i = i -1  else:  check = 0  for k = 1 to i – 1:  if Ai,k 0:  check = 1  C = create\_init(n)  C1 = create\_init(n)  swap\_row(C, k, i – 1)  swap\_col(C1, k , i – 1)  nhân 3 ma trận vuông: A = C\*A\*C1  nhân 2 ma trận vuông: P = P\*C1  End for  i = i - 1  if check 1:  for j = i to n – 1:  S = create\_init(n)  S1 = create\_init(n)  for k = 1 to i – 1:  Sk,j+1 = - Ak,j  S1k,j+1 = Ak,j  nhân 3 ma trận vuông: A = S\*A\*S1  nhân 2 ma trận vuông: P = P\*S1  // Kiểm tra hàng cuối cùng xem có phần tử 0  case2 = 0  for k = 1 to i – 1:  if Ak,n ≠ 0:  case = 1  U = create\_init(n)  UT = create\_init(n)  for j = i to n:  Uj,j = 0  Uj,j+1 = 1  UTj,j-1 = 1  Un,i = 1  UTi,n = 1  nhân 3 ma trận vuông: A = U\*A\*UT  nhân 2 ma trận vuông: P = P\*UT  i = n  End for  if case2 = 0:  // Tìm trị riêng và vector riêng của ma trận A  for k = i to n:  for j = i to n:  Fk,j = Ak,j  h = n + 1 - i //cỡ ma trận vuông Frobenius F  eigenvalue[] = solution\_eigen(F) //tìm trị riêng  m = số phần tử có trong mảng eigenvalue[]  for k = 1 to m:  lamda = eigenvalue[k]  print “trị riêng là” lamda  for j = 1 to n:  if j <= n – h:  eigenvectoY[j] = 0  else:  eigenvectoY[j] = lamda ^ (n – j)  // Nhân ma trận P với mảng eigenvectoY để tìm vector riêng của ma trận A cần tìm  for t = 1 to n:  eigenvectoX[t] = 0  for j = 1 to n:  eigenvectoX[t]+=Pt,j\*eigenvectoY[j]  for t = n to row:  eigenvectoX[t] = 0  print “vector riêng tương ứng là:”  print eigenvectoX  // Tiếp tục quá trình Danilevski với ma trận Ai  n = i – 1  p = create\_init(n)  // Kết thúc quá trình biến đổi  for k = i to n:  for j = i to n:  Fk,j = Ak,j  eigenvalue[] = solution\_eigen(F) //tìm trị riêng  m = số phần tử có trong mảng eigenvalue[]  for k = 1 to m:  lamda = eigenvalue[k]  print “trị riêng là” lamda  for j = 1 to n:  eigenvectoY[j] = lamda ^ (n – j)  // Nhân ma trận P với mảng eigenvectoY để tìm vector riêng của ma trận A cần tìm  for t = 1 to n:  eigenvectoX[t] = 0  for j = 1 to n:  eigenvectoX[t] += Pt,j \* eigenvectoY[j]  for t = n to row:  eigenvectoX[t] = 0  print “vector riêng tương ứng là:”  print eigenvectoX  Kết thúc chương trình |

1. Ưu và nhược điểm của phương pháp
2. Ưu điểm

* Độ chính xác cao
* Phương pháp Danilevski là phương pháp tìm trị riêng và vecto riêng đúng nên sai số chỉ ở các phép tính toán
* Khối lượng tính toán giảm hơn so với phương pháp tính thông thường
* Thể hiện rõ được hiệu quả của việc biến đổi ma trận tương đương từ ma trận bất kì về khối Frobenius

1. Nhược điểm

* Độ phức tạp thuật toán vẫn còn lớn
* Sai số trong tính toán còn phụ thuộc vào hàm tìm trị riêng của đa thức đặc trưng khá nhiều
* Thuật toán phức tạp => khó lập trình

\*Chú ý: ta có thể sử dụng phương pháp Danilevski để tìm giá trị riêng lớn nhất, nhỏ nhất từ đó tìm vecto kì dị