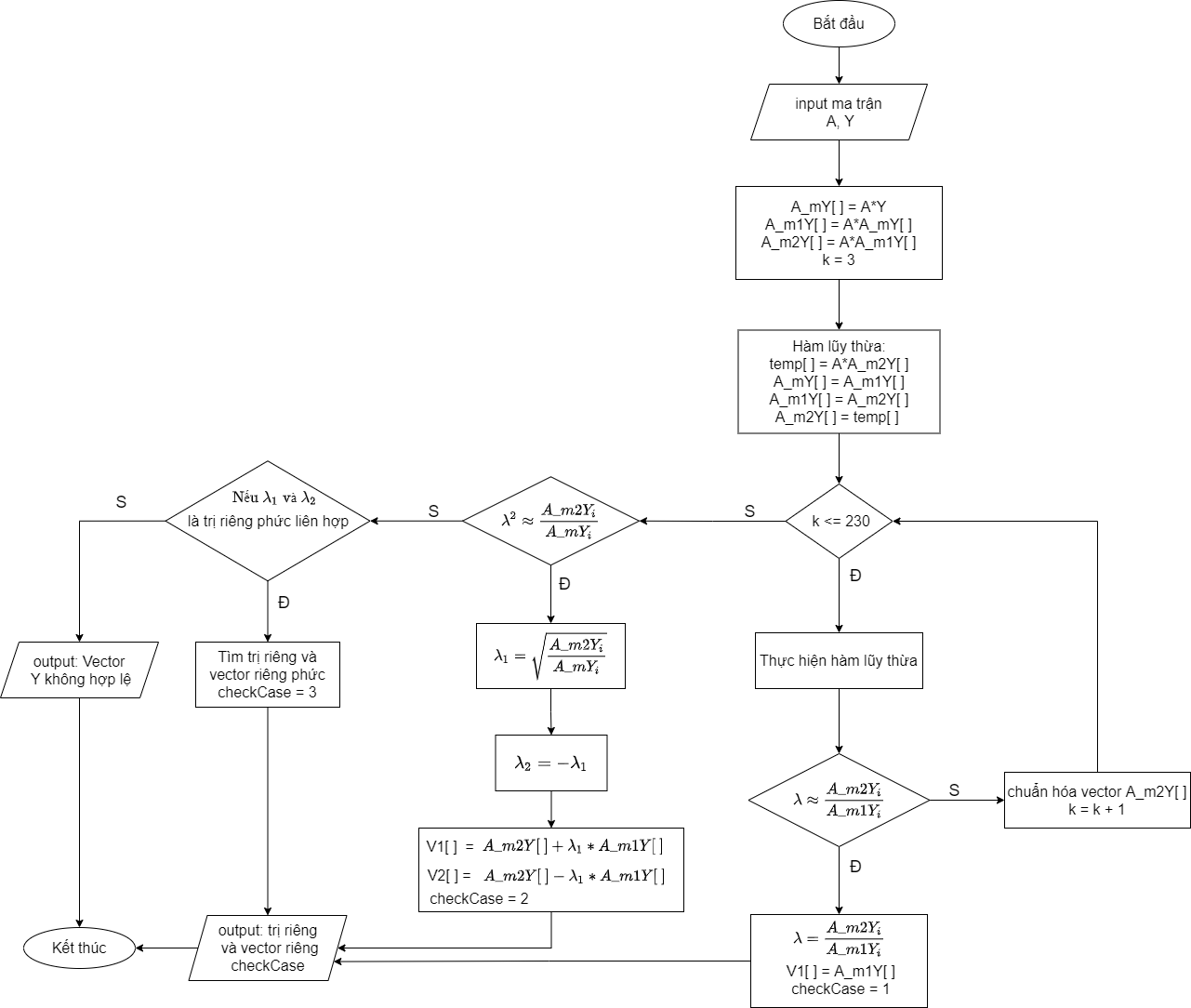
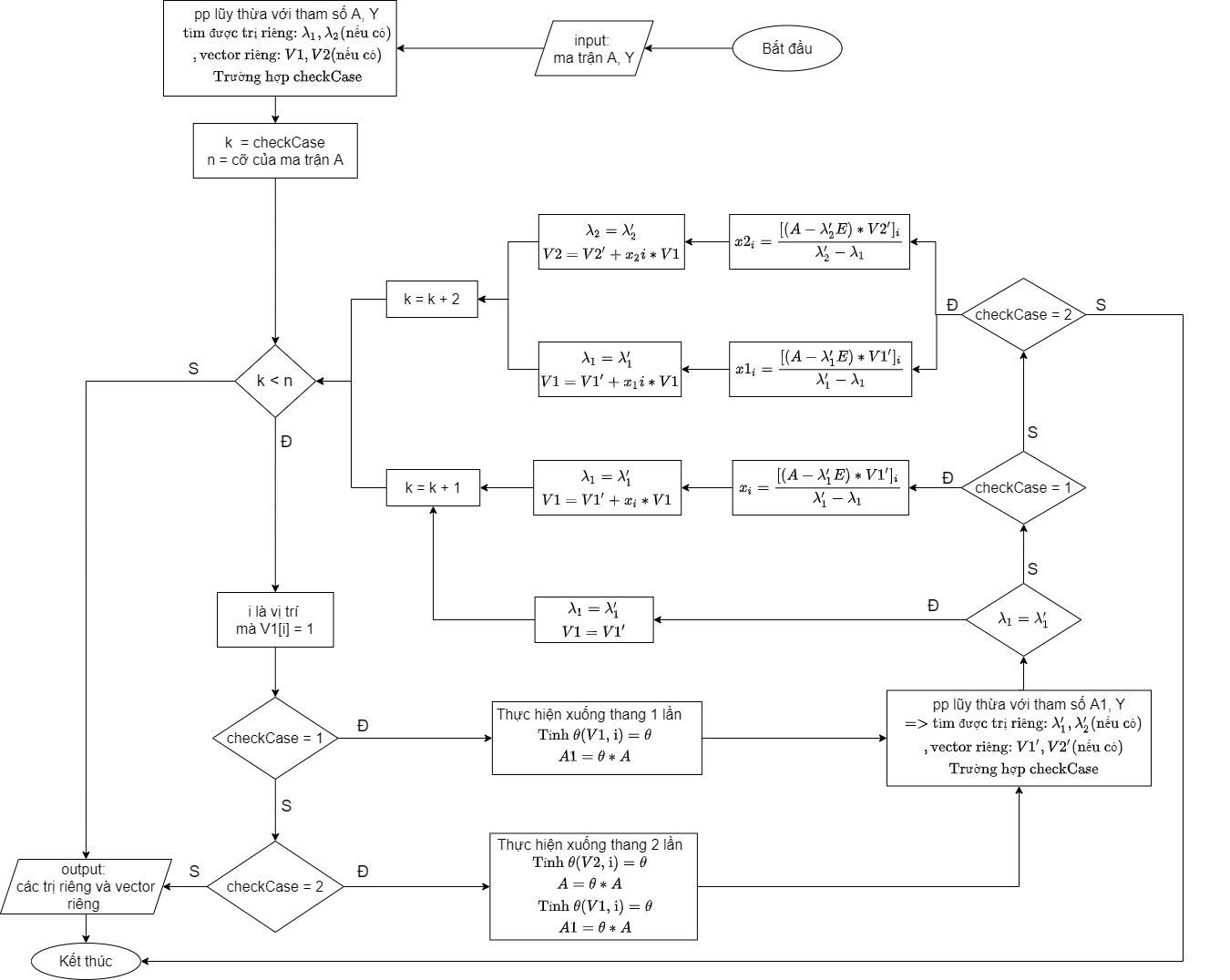
**Phương pháp lũy thừa tìm giá trị riêng trội và pp xuống thang tìm giá trị riêng trội và vector riêng trội tiếp theo**

1. Thuật toán tổng thể
2. Phương pháp lũy thừa tìm giá trị riêng trội và vector riêng



1. Phương pháp xuống thang tìm trị riêng và vector riêng tiếp th



1. Thuật toán chi tiết

|  |
| --- |
| 1. Xây dựng 1 kiểu cấu trúc ma trận gồm có:  * Thành phần ma trận: mảng 2 chiều a[][] * Số cột * Số hàng  1. Tạo ma trận theta   Input: mảng X[], vị trí i mà phần tử X[i] = 1, cỡ ma trận n  Output: ma trận theta = ma trận đơn vị E cỡ n có cột i = - X[], phần tử theta[i][i] = 0  Function create\_theta:   1. Hàm chuẩn hóa vector   Input: vector X là một mảng X[], kích cỡ vector n  Output: Vector X đã được chuẩn hóa, và trả về vị trị phần tử = 1  Function chuanHoa:  Mô tả:   * Tìm giá vị trí i phần tử có giá trị tuyệt đối lớn nhất trong mảng X * Chia tất cả các phần tử của mảng X cho phần tử tại chỉ số i  1. Hàm lũy thừa   Input: ma trận A, 3 mảng 1 chiều A\_mY, A\_m1Y, A\_m2Y lần lượt chứa 3 vector AmY, Am+1Y, Am+2Y  Output: 3 mảng 1 chiều A\_mY, A\_m1Y, A\_m2Y  Function luyThua:  n = số hàng ma trận A  for i = 0 to n – 1:  temp[i] = 0  for j = 0 to n – 1:  temp[i] = temp[i] + Aij \* A\_m2Y[j]  //Thực hiện gán lại giá trị vào 3 mảng A\_mY, A\_m1Y, A\_m2Y  for i = 0 to n – 1:  A\_mY[i] = A\_m1Y[i]  A\_m1Y[i] = A\_m2Y[i]  A\_m2Y[i] = temp[i]   1. Tìm giá trị riêng trội và vector riêng trội   Input: ma trận A, ma trận Y  Output: trị riêng và vector riêng trội tương ứng, trường hợp tương ứng  Function find\_tri\_rieng\_troi:  k = 230 // giới hạn số vòng lặp tối đa  n = số hàng của ma trận A  gán mảng A\_mY = A \* Y  chuanHoa(A\_mY, n)  gán mảng A\_m1Y = A\*A\_mY  chuanHoa(A\_m1Y, n)  gán mảng A\_m2Y = A\*A\_m1Y  viTri = chuanHoa(A\_m2Y, n)  for m = 4 to 230:  gọi hàm luyThua(A, A\_mY, A\_m1Y, A\_m2Y)  tiSo = A\_m2Y[viTri] / A\_m1Y[viTri]  check = 1  for i = 0 to n – 1:  if A\_m1Y[i] 0:  if |tiSo – A\_m2Y[i] / A\_m1Y[i]| > 0.00001:  check = 0  if check = 1:  truongHop = 1  Kết thúc vòng for (End for)  chuanHoa(A\_m2Y, n)  if truongHop ≠ 1:  gọi hàm: luyThua(A, A\_myY, A\_m1Y, A\_m2Y)  tiếp tục gọi hàm: luyThua(A, A\_mY, A\_m1Y, A\_m2Y)  check = 1  tiSo = A\_m2Y[viTri] / A\_mY[viTri]  for i = 0 to n – 1:  if A\_mY[i] ≠ 0:  if |tiSo – A\_m2Y[i] / A\_mY[i]| > 0.00001:  check = 0  if check = 1:  truongHop = 2  else  truongHop = 3  if truongHop = 1:  triRieng1 = A\_m2Y[viTri] / A\_m1Y[viTri]  vector1[] = A\_m1Y[]  return truongHop  if truongHop = 2:  heSo = A\_m2Y[viTri] / A\_mY[viTri]  triRieng1 =  triRieng2 = - triRieng1  vector1[] = A\_m2Y[] + triRieng1 \* A\_m1Y[]  vector2[] = A\_m2Y[] – triRieng1 \* A\_m1Y[]  chuanHoa(vector1, n)  chuanHoa(vector2, n)  return truongHop  if truongHop = 3:  Gọi hàm: luyThua(A, A\_mY, A\_m1Y, A\_m2Y)  a = A\_m1Y[1] \* A\_mY[0] – A\_m1Y[0] \* A\_mY[1]  b = - (A\_m2Y[1] \* A\_mY[0] – A\_m2Y[0] \* A\_mY[1])  c = A\_m2Y[1] \* A\_m1Y[0] – A\_m2Y[0] \* A\_m1Y[1]  delta = b^2 – 4 \*a\*c  if a = 0 hoặc delta >= 0:  print “Vector Y không hợp lệ” => Kết thúc chương trình  else:  reLamda = - b / 2\*a  imLamda = / (2\*a)  lamda1 = reLamda + imLamda \* i  lamda2 = relamda – imLamda \* i  print “trị riêng phức: “ lamda1  print “vecto riêng tương ứng” A\_m1Y – lamda2\*A\_mY  print “trị riêng phức: “ lamda2  print “vecto riêng tương ứng” A\_m1Y – lamda1\*A\_mY  Kết thúc chương trình   1. Hàm tìm các trị riêng tiếp theo bằng phương pháp xuống thang   Input: Ma trận A, ma trận Y  Output: in ra các trị riêng, vecto riêng tương ứng, số lần lặp  Function xuong\_thang:  Gọi hàm tìm giá trị riêng trội: find\_tri\_riêng\_troi(A, Y) trả về giá trị riêng trội: triRieng1, triRieng2; vecto riêng trội tương ứng: vector1[], vector2[], trường hợp rơi vào: truongHop  if truongHop = 1:  print “trị riêng thứ 1 là: “ triRieng1  print “vector riêng tương ứng là: “ vector1[]  k = 1  if truongHop = 2:  print “trị riêng thứ 1 là: “ triRieng1  print “vector riêng tương ứng là: vector1[]  print “trị riêng thứ 2 là: “ triRieng2  print “vector riêng tương ứng là: vector2[]  k = 2  if truongHop = 3:  kết thúc chương trình  n = số hàng của ma trận A, cũng chính là số trị riêng tối đa có thể có  viTri = vị trí phần tử = 1 sau khi gọi hàm chuanHoa(vector1, n)  while k < n:  if truongHop = 1:  theta = create\_init(vector1, viTri)  A1 = theta \* A  if truongHop = 2:  theta = create\_init(vector2, viTri)  A = theta \* A  theta = create\_init(vector1, viTri)  A1 = theta \* A  triRieng1cu = triRieng1  vector1cu[] = vector1  vector2cu[] = vector2  gọi hàm tìm giá trị riêng trội: find\_tri\_riêng\_troi(A1, Y) trả về giá trị riêng trội: triRieng1, triRieng2; vecto riêng trội tương ứng: vector1[], vector2[], trường hợp rơi vào: truongHop  if triRieng1 ≠ triRieng1cu:  if truongHop = 1:  C = A – triRieng1 \* E //E là ma trận đơn vị  temp1[] = C \* vector1[]  x1i = tem1[viTri] / (triRieng1 – triRieng1cu)  vector1[] = vector1[] + x1i \* vector1cu[]  if truongHop = 2:  C = A – triRieng1 \* E //E là ma trận đơn vị  temp1[] = C \* vector1[]  x1i = tem1[viTri] / (triRieng1 – triRieng1cu)  vector1[] = vector1[] + x1i \* vector1cu[]  C = A – triRieng2 \* E //E là ma trận đơn vị  temp2[] = C \* vector2[]  x2i = tem2[viTri] / (triRieng2 – triRieng1cu)  vector1[] = vector2[] + x2i \* vector1cu[]  if truongHop = 1:  print “trị riêng 1 là” triRieng1  print “vector riêng tương ứng là : vetor1[]  k = k + 1  if truongHop = 2:  print “trị riêng 1 là” triRieng1  print “vector riêng tương ứng là: vector1[]  print “trị riêng 2 là: triRieng2  print “vector riêng tương ứng là: vector2[]  k = k + 2  A = A1  Kết thúc chương trình |

1. Ưu và nhược điểm của thuật toán
2. Ưu điểm

* Có thể tính trị riêng và vector riêng đối với ma trận vuông cấp lớn
* Linh hoạt trong việc tìm trị riêng và vector riêng. Cụ thể là nếu chỉ cần tìm trị riêng lớn nhất thì chỉ cần dùng thuật toán lũy thừa là xong không cần phải tìm toàn bộ trị riêng
* Vì là phương pháp tính gần đúng nên tốc độ tìm ra trị riêng nhanh do độ phức tạp thuật toán thấp hơn pp Danilevski
* Trong thực tế đôi khi để đơn giản ta không cần phải đi tìm đúng các trị riêng mà chỉ cần tìm gần đúng với 1 sai số epsilon nhỏ cho phép, vì tìm gần đúng là nhanh hơn và ít phức tạp hơn rất nhiều so với tìm đúng

1. Nhược điểm

* Tốc độ hội tụ không ổn định
* Một số ma trận không thể áp dụng phương pháp này do khi xấp xỉ nhiều lần, tính hội tụ không còn đúng
* Dễ bị tràn số nếu không chuẩn hóa vector
* Vẫn chưa xử lý được trường hợp trị riêng phức liên hợp để xuống thang

1. Ứng dụng

* Được ứng dụng trong việc xử lí ảnh, nén ảnh khi ta cần phải tìm các trị riêng và vector riêng nhanh chóng để biến đổi ảnh