Actividad A_3.

Algebra Lineal, Métodos Iterativos

Ignacio Sica

24/04/21

Contents

Programe los siguientes algoritmos:

Algoritmo de Jacobi para la solución de sistemas de ecuaciones lineales.

Algoritmo de Sobre relajación sucesiva (SOR) para la solución de sistemas de ecuaciones lineales.

Algoritmo de gradiente conjugado para la solución de sistemas de ecuaciones lineales (Tabla 3.1; Walter 2014)

```
load datos1

a = {a1,a2,a3,a4,a5};
b = {b1,b2,b3,b4,b5};
```

Ejercicio n#1

Utilizando el archivo datos1 resolver los sistemas de ecuaciones $a_i x_i = b_i$, para i=1,2,...,5, con los algoritmos de Jacobi, SOR con ? =0.5,1,1.5 y Gradiente Conjugado. Compare los errores relativos con respecto a los vectores bi y el tiempo de solución necesario. Comente sus resultados.

```
disp("Ejercicio 1:"), disp("")

tol      = 0.0000001;
max_iter = 4500;

for i = 1 : 5
  printf("a%d: ", i), disp("")
  ej1(a{i}, b{i}, tol, max_iter);
  disp("")
endfor
```

```
Ejercicio 1:

a1:
Jacobi:
  Time elapsed: 0.000219107s
  Error: 6.99504e-08
  Total iterations: 8
SOR 0.5:
  Time elapsed: 0.00261188s
  Error: 6.27048e-08
  Total iterations: 31
SOR 1.0:
  Time elapsed: 0.000573158s
  Error: 1.83432e-08
  Total iterations: 7
SOR 1.6:
  Time elapsed: 0.00397301s
  Error: 6.31762e-08
  Total iterations: 33
Gradiente Conjugado:
  Time elapsed: 0.0136571s
  Error: 3.24057e-18
  Total iterations: 5

a2:
Jacobi:
  Time elapsed: 0.00141001s
  Error: 5.22151e-08
  Total iterations: 17
SOR 0.5:
  Time elapsed: 0.049155s
  Error: 8.48051e-08
  Total iterations: 47
SOR 1.0:
  Time elapsed: 0.013808s
  Error: 5.34291e-08
  Total iterations: 13
SOR 1.6:
  Time elapsed: 0.0344951s
  Error: 6.69823e-08
  Total iterations: 33
Gradiente Conjugado:
  Time elapsed: 0.019558s
  Error: 8.44988e-10
  Total iterations: 9

a3:
Jacobi:
  Time elapsed: 0.0528769s
  Error: 9.55273e-08
  Total iterations: 164
SOR 0.5:
  Time elapsed: 0.910161s
```

```
  Error: 9.97661e-08
  Total iterations: 428
SOR 1.0:
  Time elapsed: 0.284088s
  Error: 9.49895e-08
  Total iterations: 142
SOR 1.6:
  Time elapsed: 0.0795562s
  Error: 6.59567e-08
  Total iterations: 32
Gradiente Conjugado:
  Time elapsed: 0.0174699s
  Error: 9.62453e-08
  Total iterations: 14

a4:
Jacobi:
  Time elapsed: 3.84483s
  Error: 9.97143e-08
  Total iterations: 2373
SOR 0.5:
  Time elapsed: 63.1061s
  Error: 8.04135e-05
  Total iterations: 4500
SOR 1.0:
  Time elapsed: 38.2843s
  Error: 9.95356e-08
  Total iterations: 2597
SOR 1.6:
  Time elapsed: 9.03455s
  Error: 9.96408e-08
  Total iterations: 671
Gradiente Conjugado:
  Time elapsed: 0.023052s
  Error: 1.76127e-08
  Total iterations: 14

a5:
Jacobi:
  Time elapsed: 25.0019s
  Error: 1.39013e-06
  Total iterations: 4500
SOR 0.5:
  Time elapsed: 286.458s
  Error: 0.0711152
  Total iterations: 4500
SOR 1.0:
  Time elapsed: 287.452s
  Error: 0.000519417
  Total iterations: 4500
SOR 1.6:
  Time elapsed: 163.867s
  Error: 9.96586e-08
  Total iterations: 2504
Gradiente Conjugado:
  Time elapsed: 0.06253s
  Error: 1.77779e-08
  Total iterations: 13
```

A modo de comienzo de analisis vale la pena descatar como a medida que aumenta el tamano de las matrices la cantidad de computo necesario para resolver los sistemas de ecuaciones mediante todos los metodos de aproximacion trabajados aumenta considerablemente. A partir de las diferentes pruebas se pueden destacar varios puntos significativos que muestran las diferencias entre los algoritmos: **(1)** es que el algoritmo del gradiente conjugado es extremadamente rapido en comparacion con los otros dos, no solo necesita de un numero de iteraciones insignificantes con respecto a los otros sino que tambien alcanza una precision que el algoritmo de SOR y jacobi en ocaciones ni siquiera alcanza, es muy visible ahora como es un algoritmo tan influyente en el mundo actual y como este ayudo al progreso de la ciencia habilitando a cientificos a resolver sistemas en cuestion de minutos que antes podrian significar dias de computacion. **(2)** se nota como a medida que aumenta el tamano de las matrices es cada vez mas complicado alcanzar la tolerancia deseada, en todos los algoritmos es igual. **(3)** dependiendo de la relajacion en el algoritmo de SOR se pueden obtener resultados muy distintos. En el caso de que de la bajo relajacion se puede notar como el algoritmo necesita un gran numero de iteraciones y en el caso de las dos matrices mas grandes ni siquiera con 4500 iteraciones logra la tolerancia deseada. A medida que aumenta el omega, es decir, se sobre relaja, los resultados son muy distintos convergiendo hacia la solucion de una manera mucho mas rapida y asi conseguir una tolerancia mejor en menos iteraciones(decidi cambiar el omega de 1.5 por 1.6 ya que estuve realizando pruebas y con 1.6 se convergia de manera mas rapido hacia el resultado). **(4)** Se puede notar como el algoritmo de jacobi necesita de mas iteraciones para llegar a la tolerancia deseada que el algoritmo de SOR con una omega mayor a uno pero sin embargo necesita de menos tiempo para resolverlo, esto se debe a la naturaleza de cada algoritmo siendo la iteracion en el algoritmo de SOR mas costosa computacionalmente hablando que la de jacobi.

Ejercicio n#2

Utilizando el archivo datos2 resolver los sistemas de ecuaciones $a_i x_i = b_i$, para i=3,...,14, utilizando la descomposición PLU. Calcule los errores relativos con respecto a los vectores bi, los errores relativos con respecto a la solución verdadera en todos los casos que es un vector de unos en todas las entradas, calculo los números de condición de los sistemas (que coincidan con la definición del libro) y concluya si los resultados obtenidos se pueden explicar con esta información. Justifique sus conclusiones.

```
load datos2

a = {a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14};
b = {b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14};

disp("Ejercicio 2:"), disp("")

for i = 3 : 13
  printf("a%d: ", i), disp("")
  ej2(a{i}, b{i});
  disp("")
endfor
```

```
Ejercicio 2:

a3:
  PLU
  Time elapsed: 0.00493908s
  Error 1: 0
  Error 2: 6.313e-16
  Cifras significativas: 14
  Número de condición: 524.057

a4:
  PLU
  Time elapsed: 0.00357604s
  Error 1: 0
  Error 2: 1.51191e-13
  Cifras significativas: 12
  Número de condición: 15513.7

a5:
  PLU
  Time elapsed: 0.004807s
  Error 1: 1.62117e-16
  Error 2: 3.70729e-13
  Cifras significativas: 12
  Número de condición: 476607

a6:
  PLU
  Time elapsed: 0.0058682s
  Error 1: 8.964e-17
  Error 2: 2.48618e-10
  Cifras significativas: 9
  Número de condición: 1.49511e+07

a7:
  PLU
  Time elapsed: 0.00721979s
  Error 1: 6.46991e-17
  Error 2: 1.29741e-08
  Cifras significativas: 7
  Número de condición: 4.75367e+08

a8:
  PLU
  Time elapsed: 0.00868487s
  Error 1: 6.55824e-17
  Error 2: 1.19995e-07
  Cifras significativas: 6
  Número de condición: 1.52576e+10

a9:
  PLU
  Time elapsed: 0.0103052s
  Error 1: 8.29979e-17
  Error 2: 1.46937e-05
  Cifras significativas: 4
  Número de condición: 4.93154e+11

a10:
  PLU
  Time elapsed: 0.011987s
  Error 1: 1.20196e-16
  Error 2: 0.000267894
  Cifras significativas: 3
  Número de condición: 1.60244e+13
```

```
a11:
  PLU
  Time elapsed: 0.013849s
  Error 1: 1.24396e-16
  Error 2: 0.000506984
  Cifras significativas: 2
  Número de condición: 5.22268e+14

a12:
  PLU
  Time elapsed: 0.01577s
  Error 1: 1.20395e-16
  Error 2: 0.148813
  Cifras significativas: 0
  Número de condición: 17514731907091464

a13:
  PLU
  Time elapsed: 0.017817s
  Error 1: 2.05618e-16
  Error 2: 7.2455
  Cifras significativas: -2
  Número de condición: 3344143497338461184
```

Con los resultado de este ejercicio se pueden sacar resoluciones muy intersantes en cuanto a los errores con respecto a la pre-imagen e imagen de las funciones. A medida que las funciones aumentan se puede notar como el error con respecto a los vectores bi disminuye y en todos los casos uno podria decir que se trata de un resultado muy confiable pero, cuando se analiza el error con respecto a la solucion verdadera del sistema se puede observar como el este segundo error es mucho mas grande y en el caso de las matrices mas grandes cuenta con **0** cifras significativas. Con respecto al tiempo necesario para resolver los sitemas se puede destacar que es muy chico no superando el segundo en ningun caso. Otro elemento a destacar del ejercicio es nuevamente la relevancia del numero de condicion, se puede observar como por cada orden de magnitud del numero de condicion disminuye en uno la cantida de digitos significativos. Con un numero de condicion lo suficientemente grande se llega al punto de no obtener ninguna cifra significativa ya que la precicion con la que se trabaja no es suficiente. Con esto en mente se puede concluir que los algoritmos de aproximacion es muy superiror al algoritmo de resolucion mediante plu ya que se pueden obtener resultados con una tolerancia mucho menor y en matrices significativamente mas grandes.

Bibliografia

Eric Walter, Springer, Numerical Methods and Optimization

Richard Khoury & Douglas Wilhelm Harder, Springer Numerical Methods and Modelling for Engineering

Gauss–Seidel method, Wikipedia, (https://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method)

Norm (mathematics), Wikipedia (https://en.wikipedia.org/wiki/Norm_(mathematics))

Javier Segura, Universidad de Cantabria, Introduccion al analisis numerico (https://personales.unican.es/segurajj/intro.pdf)

Errors for Linear Systems (http://terpconnect.umd.edu/~petersd/460/linsysterrn.pdf)

Inv Function (https://octave.sourceforge.io/octave/function/inv.html)

Jacobi

```
function [X_sol,err,total_iter] = jacobi (A, b, X0, tol, Max_iter)
  err = 10;
  contador = 0;
  D = diag(diag(A));
  M1 = A-D;
  Dinv = diag( 1./diag(A) );
  while err>tol && contador<Max_iter
    X1 = Dinv*(-M1*X0 + b);
    err = norm(X1-X0);
    contador = contador + 1;
    X0 = X1;
  endwhile
  X_sol = X1;
  total_iter = contador;
endfunction
```

SOR

```
function [X_sol,err,total_iter] = SOR (A, b, X, om, tolerance, max_iter)

  D = diag(diag(A));
  L = tril(A, -1);
  U = triu(A, +1);

  err = 10;
  counter = 0;

  j = D + om * L;
  k = om * b;
  i = (om * U + (om -1) * D);

  while err > tolerance && counter < max_iter

    X = forward(j, k - i * X);
    err = norm(A * X - b) / norm(b);
    counter = counter + 1;

  endwhile

  X_sol = X;
  total_iter = counter;
```

```
endfunction
```

Gradiente conjugado

```
function [x_sol, err, total_iter] = GC (A, b, x, tolerance, max_iter)

  err = 10;
  counter = 0;

  r   = b - A * x;
  d   = r;
  sig_0 = normest(r)^2;
  k     = 0;

  while normest(r) > tolerance && k < max_iter

    sig_a = d' * (A * d);
    lam = sig_0 / sig_a;
    x = x + lam * d;
    r = r - lam * A * d;
    sig_1 = normest(r)^2;
    bet = sig_1 / sig_0;
    d = r + bet * d;
    sig_0 = sig_1;
    k = k + 1;

  endwhile

  x_sol = x;
  err = normest(r);
  total_iter = k;

endfunction
```

SustitucionAdelante

```
function [y] = SustitucionAdelante (L, b)
  [mL,nL] = size(L);
  [mb,nb] = size(b);
  y = 0*b;

  if (L(1,1)==1)
    y(1) = b(1);
  else
    y(1) = b(1)/L(1,1);
  endif
  for ii=2:mL
    y(ii) = b(ii)-L(ii,1:ii-1)*y(1:ii-1);
    pivote = L(ii,ii);
    if (pivote ~=1)
      y(ii) = y(ii)/pivote;
    endif
  endfor
endfunction
```

forward

```
function x = forward (A, b)
  n = length(b);
  x(1,1) = b(1)/A(1,1);
  for i = 2:n
      x(i,1)=(b(i)-A(i,1:i-1)*x(1:i-1,1))./A(i,i);
  end
endfunction
```

SustitucionAtras

```
function [y,msg] = SustitucionAtras (U, b)
  [mU,nU] = size(U);
  [mb,nb] = size(b);

  if mU ~= nU
    msg = 0;
    y = inf;
    disp('U no es cuadrada')
    return; # fin de la funcion
  elseif (mU ~= mb) || nb ~= 1
    msg = 0;
    y = inf;
    disp('U y b son de dimensiones incompatibles')
    return; # fin de la funcion
  endif
  for ii=1:mU
    if abs(U(ii,ii)) < eps
      msg = 0;
      y = inf;
      disp('U tiene pivote menor que eps')
      return; # fin de la funcion
    endif
  endfor

  ## Inicio del algoritmo
  y = 0*b;

  if (U(end,end)==1)
    y(end) = b(end);
  else
    y(end) = b(end)/U(end,end);
  endif
  for ii=1:nU-1
    y(nU-ii) = b(nU-ii)-U(nU-ii,nU-ii:end)*y(nU-ii:end);
    pivote = U(nU-ii,nU-ii);
    if (pivote ~=1)
      y(nU-ii) = y(nU-ii)/pivote;
    endif
  endfor

  msg = 1;
 % disp('Algoritmo finalizo normalmente');

endfunction
```

Codigo ej1

```
function ej1 (A, b, tol, Max_iter)
  x = zeros(size(b));
```

```
  disp("Jacobi:")
  tic();
  [x_sol_jacobi,err,total_iter] = jacobi (A, b, x, tol, Max_iter);
  time = toc ();
  printf("  Time elapsed: %ds", time), disp("")
  printf("  Error: %d", err), disp("")
  printf("  Total iterations: %d", total_iter), disp("")

  disp("SOR 0.5:")
  tic();
  [x_sol_sor_05,err,total_iter] = SOR(A, b, x, 0.5, tol, Max_iter);
  time = toc ();
  printf("  Time elapsed: %ds", time), disp("")
  printf("  Error: %d", err), disp("")
  printf("  Total iterations: %d", total_iter), disp("")

  disp("SOR 1.0:")
  tic();
  [x_sol_sor_10,err,total_iter] = SOR(A, b, x, 1.0, tol, Max_iter);
  time = toc ();
  printf("  Time elapsed: %ds", time), disp("")
  printf("  Error: %d", err), disp("")
  printf("  Total iterations: %d", total_iter), disp("")

  disp("SOR 1.6:")
  tic();
  [x_sol_sor_15,err,total_iter] = SOR(A, b, x, 1.6, tol, Max_iter);
  time = toc ();
  printf("  Time elapsed: %ds", time), disp("")
  printf("  Error: %d", err), disp("")
  printf("  Total iterations: %d", total_iter), disp("")

  disp("Gradiente Conjugado:")
  tic();
  [x_sol_gc,err,total_iter] = GC(A, b, x, tol, Max_iter);
  time = toc ();
  printf("  Time elapsed: %ds", time), disp("")
  printf("  Error: %d", err), disp("")
  printf("  Total iterations: %d", total_iter), disp("")
endfunction
```

Codigo ej2

```
function ej2 (A, b)

  disp ("  PLU")
  tic();
  y = sem_plu(A, b);
  time = toc ();
  printf("  Time elapsed: %ds", time), disp("")
  error = norm(A * y - b) / norm(b);
  printf("  Error 1: %d", error),
  disp ("")
  sol = ones(size(y),1);
  error = norm(y - sol) / norm(sol);
  printf("  Error 2: %d", error),
  disp ("")
  Sd=floor(-log10(error/0.5));
  printf("  Cifras significativas: %d", Sd),
  disp("")
  nc = cond(A);
  printf("  Número de condición: %d", nc),
```

```
  disp("")

endfunction
```

Published with GNU Octave 6.2.0