Universidad Nacional Autónoma de México UNAM IIMAS

Computo de Alto Rendimiento en lenguajes de alto nivel

Presentación Miniproyecto 2

Dr. Oscar Esquivel Flores

Juan Carlos López Núñez

Objetivos de Proyecto

- **Propósito del repositorio**
- El propósito de este repositorio es el desarrollar una serie de programas en lenguajes de alto nivel
- para la resolución se sistemas de ecuaciones lineales de la forma Ax = b por medio de los sig. métodos iterativos

- **Jacobi**
- **Gauss Seidel**
- **Sobre relajación sucesiva SOR**
- **Gradiente conjugado**

Descripción del proyecto

- Para este proyecto se busca la implementación de distintos métodos de solución de
- sistemas de ecuaciones en los lenguajes de alto nivel Julia y Python.

Introducción

- El uso de métodos iterativos para la solución de sistemas de ecuaciones de la forma Ax=b. Para el caso de los sistemas de ecuaciones de mayor tamaño estas estrategias permiten la resolución de sistemas de ecuaciones.
- Para el caso en el cual se tienen grandes sistemas de ecuaciones se utilizan matrices dispersas las cuales la mayoria de sus elementos son ceros, de esta manera se reduce la memoria almacenada de las matrices lo cual es un aspecto que se deb considerar en el caso de grandes sistemas de ecuaciones.

- Los métodos iterativos se basan en una serie de aproximaciones sucesivas hasta poder satisfacer un criterio de convergencia.

Contexto

Uno de los métodos numéricos estudiados es el método de Jacobi

Método de Gauss Seidel
Este método se basa en el método de Jacobi por lo que se puede calcular usando los mas recientes cálculos. En este método se utiliza siempre el ultimo valor para el próximo calculo.

El método de sobre relajación sucesiva.
Esta basado en Gauss Seidel para poder tener una convergencia mas rápida

Constante omega
Para la constante omega > 1 denominado factor de relajación. Este método en cada una de las iteraciones usa el valor de x del paso anterior en el lado derecho.

La relación con el método de Gauss Seidel se tome de omega = 1 se tiene el método de Gauss Seidel como un caso particular de SOR. Para los otros valores con el método de SOR.

Método de Jacobi Julia

Funcion de Jacobi

In [57]:
```
using Printf
Base.show(io::IO, f::Float64) = @printf(io, "%.3e", f)
using LinearAlgebra
using SparseArrays
```

In [58]:
```
function jacobi(mat::Array{Float64}, rhs::Array{Float64},
                sol::Array{Float64},
                maxit::Int=100,tol::Float64=1.0e-8)
    nbrows, nbcols = size(mat)
    result = deepcopy(sol)
    numit = 0; nrmdx = 1
    while numit < maxit
        numit = numit + 1
        deltax = rhs - mat * result
        for i = 1:nbrows
            deltax[i] = deltax[i] / mat[i,i]
        end
        result = result + deltax
        nrmdx = norm(deltax)
        strdx = @sprintf("%.2e", nrmdx)
        println("||dx|| = $strdx")
        if norm(deltax) <= tol
            return (result, numit, nrmdx, false)
        end
    end

    return (result, numit, nrmdx, true)
end
```

Out[58]:
```
jacobi (generic function with 3 methods)
```

Método de Jacobi

```
In [60]: # include("jacobi.jl")

import Random #to fix the seed of the random numbers

function jacobiTest()
    print("Give the dimension: ")
    line = readline(stdin)
    dim = parse(Int, line)
    Random.seed!(123);
    mat = rand(dim, dim)
    # ------------------------------------------------
    #mat = sprand(dim,dim,2/5)
    #mat=Matrix(mat)
    # -----------------------------------------
    #matriz dispersa
    brow = randperm!(MersenneTwister(1234), Vector{Int}(undef, dim))
    bcol = randperm!(MersenneTwister(1234), Vector{Int}(undef, dim))
    data = randperm!(MersenneTwister(1234), Vector{Int}(undef, dim))

    sparseA = sparse(brow, bcol, data, dim, dim)
    #definicion de matriz dispersa A
    sparseA =Matrix(sparseA)
    mat = float(sparseA)

    # ----------------------------------------


    # --------------------------------------------------

    # https://en.wikipedia.org/wiki/Diagonally_dominant_matrix
    #make the matrix diagonally dominant
    for i=1:dim
        mat[i,i] = 100*mat[i,i]
    end
    sol = ones(dim, 1)
    noise = (1.0e-4)*rand(dim, 1)
    rhs = mat*sol
    wrk = sol + noise
    println("A random matrix: ")
    show(stdout, "text/plain", mat); println("");
    sol, numit, nrmdx, fail = jacobi(mat, rhs, wrk)
    println("The solution after ", numit, " iterations: ")
    for i=1:dim
        strsol = @sprintf("%.16f", sol[i])
        println(i, " : $strsol")
    end
    print("Estimated forward error: ", nrmdx)

    if fail
        println(" failed.")
    else
        println(" succeded.")
    end
end

@time jacobiTest()
```

Pruebas con la función de Jacobi

20

```
 0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.000
e+00  0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.000e+00
 0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.000
e+00  0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.000e+00
||dx|| = 3.17e-04
||dx|| = 3.62e-16
The solution after 2 iterations:
1 : 1.0000000000000000
2 : 1.0000000000000000
3 : 1.0000000000000000
4 : 1.0000000000000000
5 : 1.0000000000000000
6 : 1.0000000000000000
7 : 1.0000000000000000
8 : 1.0000000000000000
9 : 1.0000000000000000
10 : 1.0000000000000000
11 : 1.0000000000000000
12 : 1.0000000000000000
13 : 1.0000000000000000
14 : 1.0000000000000000
15 : 1.0000000000000000
16 : 1.0000000000000000
17 : 1.0000000000000000
18 : 1.0000000000000000
19 : 1.0000000000000000
20 : 1.0000000000000000
Estimated forward error: 3.616e-16 succeded.
  3.797590 seconds (26.28 k allocations: 1.455 MiB)
```

40

```
26 : 1.0000000000000000
27 : 1.0000000000000000
28 : 1.0000000000000000
29 : 1.0000000000000000
30 : 1.0000000000000000
31 : 1.0000000000000000
32 : 1.0000000000000000
33 : 1.0000000000000000
34 : 1.0000000000000000
35 : 1.0000000000000000
36 : 1.0000000000000000
37 : 1.0000000000000000
38 : 1.0000000000000000
39 : 1.0000000000000000
40 : 1.0000000000000000
Estimated forward error: 4.697e-16 succeded.
 10.400884 seconds (101.82 k allocations: 5.429 MiB)
```

50

```
40 : 1.0000000000000000
41 : 1.0000000000000000
42 : 1.0000000000000000
43 : 1.0000000000000000
44 : 1.0000000000000000
45 : 1.0000000000000000
46 : 1.0000000000000000
47 : 1.0000000000000000
48 : 1.0000000000000000
49 : 1.0000000000000000
50 : 1.0000000000000000
Estimated forward error: 2.652e-16 succeded.
 10.179225 seconds (158.34 k allocations: 8.437 MiB)
```

80

```
72 : 1.0000000000000000
73 : 1.0000000000000000
74 : 1.0000000000000000
75 : 1.0000000000000000
76 : 1.0000000000000000
77 : 1.0000000000000000
78 : 1.0000000000000000
79 : 1.0000000000000000
80 : 1.0000000000000000
Estimated forward error: 6.258e-16 succeded.
 26.332468 seconds (402.78 k allocations: 21.485 MiB, 0.02% gc time)
```

100

```
92 : 1.0000000000000000
93 : 1.0000000000000000
94 : 1.0000000000000000
95 : 1.0000000000000000
96 : 1.0000000000000000
97 : 1.0000000000000000
98 : 1.0000000000000000
99 : 1.0000000000000000
100 : 1.0000000000000000
Estimated forward error: 7.902e-16 succeded.
 32.692264 seconds (627.69 k allocations: 33.406 MiB, 0.05% gc time)
```

Método de Gauss Seidel Julia

20

```
11 : 1.0000692208662054
12 : 1.0000136551475138
13 : 1.0000032096673352
14 : 1.0000350545821459
15 : 1.0000930332376383
16 : 1.0000959433599408
17 : 1.0000581912342388
18 : 1.0000311447500705
19 : 1.0000121147520518
20 : 1.0000204529817320
Estimated forward error: 2.423e-04 failed.
  2.824736 seconds (27.68 k allocations: 1.547 MiB)
```

40

```
31 : 1.0000891376975714
32 : 1.0000711038549801
33 : 1.0000360582900318
34 : 1.0000259562825942
35 : 1.0000390013031843
36 : 1.0000461862227734
37 : 1.0000934015587901
38 : 1.0000753277849992
39 : 1.0000729599728300
40 : 1.0000162380425148

Estimated forward error: 3.677e-04 failed.
  9.390113 seconds (103.29 k allocations: 5.598 MiB)
```

50

```
41 : 1.0000637077255792
42 : 1.0000991405136186
43 : 1.0000383548517595
44 : 1.0000618206502669
45 : 1.0000484289326814
46 : 1.0000599701927744
47 : 1.0000453693365015
48 : 1.0000324903420243
49 : 1.0000581912342388
50 : 1.0000311447500705
Estimated forward error: 4.097e-04 failed.
 10.415585 seconds (159.83 k allocations: 8.556 MiB, 0.08% gc time)
```

80

```
70 : 1.0000437964760285
71 : 1.0000231468262133
72 : 1.0000647142779366
73 : 1.0000681639183140
74 : 1.0000122394199855
75 : 1.0000648813278765
76 : 1.0000587629851110
77 : 1.0000377373211393
78 : 1.0000240446811441
79 : 1.0000222149434654
80 : 1.0000509696476905
Estimated forward error: 5.030e-04 failed.
 22.152685 seconds (404.16 k allocations: 21.580 MiB)
```

100

```
91 : 1.0000047981457638
92 : 1.0000426778850118
93 : 1.0000972793987020
94 : 1.0000928161619076
95 : 1.0000077528627520
96 : 1.0000538916761665
97 : 1.0000581912342388
98 : 1.0000311447500705
99 : 1.0000121147520518
100 : 1.0000204529817320
Estimated forward error: 5.606e-04 failed.
 31.762318 seconds (629.31 k allocations: 33.547 MiB, 0.02% gc time)
```

Método de Sobre relajación sucesiva SOR

20

```
11 : 5733.7319338225624961
12 : 1131.8916589166390168
13 : 266.8181476125612335
14 : 2904.1494910537721807
15 : 7705.8243044487535371
16 : 7946.8347391443903689
17 : 4820.2801545917754993
18 : 2580.3451178457025890
19 : 1004.3192267851676434
20 : 1694.8745199578634129
Estimated forward error: 1.840e+04 failed.
  9.998820 seconds (27.74 k allocations: 1.548 MiB)
```

40

```
38 : 6239.4945788741188153
39 : 6043.3971709491106594
40 : 1345.8017912680629706
Estimated forward error: 2.792e+04 failed.
  7.978148 seconds (103.47 k allocations: 5.604 MiB)
```

50

```
46 : 4967.6098972227164268
47 : 3758.3965544668553775
48 : 2691.7843179483315907
49 : 4820.2801546086575399

50 : 2580.3451178426657862
Estimated forward error: 3.110e+04 failed.
 10.636301 seconds (159.82 k allocations: 8.609 MiB)
```

80

```
73 : 5646.1976502317884297
74 : 1014.6439725464369985
75 : 5374.3401590098428642
76 : 4867.6314037990714496
77 : 3126.3285006055316444
78 : 1992.3317904017717410
79 : 1840.7966219726231429
80 : 4222.2029838171010852
Estimated forward error: 3.819e+04 failed.
 22.336392 seconds (404.18 k allocations: 21.581 MiB, 0.06% gc time)
```

Pruebas de rendimiento

100

```
91 : 398.3727135801129862
92 : 3535.4959935767860770
93 : 8057.4827632143587834
94 : 7687.8465320889226859
95 : 643.0763898547307917
96 : 4464.1994636982881275
97 : 4820.2801545929232816
98 : 2580.3451178439031537
99 : 1004.3192267629767684
100 : 1694.8745200026178281
Estimated forward error: 4.256e+04 failed.
 41.596646 seconds (629.41 k allocations: 33.555 MiB)
```

Gradiente conjugado

Gradiente conjugado

In [74]:
```
using Printf
using LinearAlgebra
```

In [75]:
```
function CGM(A::Array{Float64,2},b::Array{Float64,1},
                x0::Array{Float64,1},
                maxit::Int64=10,tol::Float64=1.0e-8,
                verbose=true)
    sol = deepcopy(x0)
    r = b - A*sol
    p = deepcopy(r)
    if verbose
        println("norm(r)     alpha        beta")
    end
    for i=1:maxit
        res = norm(r)
        if verbose
            sres = @sprintf("%.2e", res)
            print("$sres")
        end
        if res < tol
            if verbose
                println("  succeded after ", i, " steps")
            end
            return (sol, res, i, false)
        end

        alpha = (transpose(r)*r)/(transpose(p)*A*p)
        if verbose
            salpha = @sprintf("%.4e", alpha)
            print("  $salpha")
        end
        sol = sol + alpha*p
        r1 = r - alpha*A*p
        beta = (transpose(r1)*r1)/(transpose(r)*r)
        if verbose
            sbeta = @sprintf("%.4e", beta)
            println("   $sbeta")
        end
        p = r1 + beta*p
        r = r1
    end
    return (sol, norm(r), maxit, true)
end
```

Función de Jacobi

```
In [55]: import numpy as np
         import math
```

```
In [56]:
         #definicion de la funcion de jacobi
         def jacobi(A, b, x0, eps=1e-10, n = 500):   #valores por defecto en la funcion de jacobi 500 iteraciones
             D = np.diag(np.diag(A))
             LU = A - D
             x = x0
             for i in range(n):
                 D_inv = np.linalg.inv(D)
                 xTemp = x
                 x = np.dot(D_inv, np.dot(-(LU), x) + b)    #jacobi
                 print('paso:', i, '- x:', x)
                 if np.linalg.norm(x - xTemp) < eps:    #norm nos da la norma de un vector
                     return x   #si converge regresa el valor calculado
             return x    #si no llego a convergencia regresa x
```

```
In [69]: # valores de prueba
#matriz A
A = np.array([
    [5, 2, 1, 1],
    [2, 6, 2, 1],
    [1, 2, 7, 1],
    [1, 1, 2, 8]
])

#matriz b transpuesta
b = np.array([29, 31, 26, 19])
x0 = np.random.rand(4)
# --------------------------------------
import numpy as np
from scipy import sparse

n = 100   #valor de prueba
matrizA = np.random.uniform(size=(n, n))
#print(X)

#X[X < 0.7] = 0   #matriz dispersa
A = np.array(matrizA)
b = np.random.uniform(n)
x0 = np.random.rand(n)

b = np.array(n)
x0 = np.array(n)

# -------------------------------------
#siendo n el numero de pasos
x = jacobi(A, b, x0, n = 10)     #si no ingresamos los ultimos valores, usa los valores por defecto la funcion
# x = jacobi(A, b, x0, 10**(-14), 500)
print("--------------------")
print('x:', x)
print("--------------------")
#print('b calculado:', np.dot(A,x))
#print('b verdadero:', b)
#solucion de numpy
#print('solucion de numpy:', np.linalg.solve(A,b))
```

Gauss Seidel

```
In [70]: #definicion de la funcion de Gauss Seidel
def gaussSeidel(A, b, x0, eps=1e-10, n = 500):   #valores por defecto en la funcion de jacobi
    D = np.diag(np.diag(A)); U = np.triu(A, k = 1); L = np.tril(A, k = -1)
    LU = A - D
    x = x0
    #D = np.diag(np.diag(A))

    for i in range(n):
        #D_inv = np.linalg.inv(D)
        LD_inv = np.linalg.inv(L+D)
        xTemp = x
        #x = np.dot(D_inv, np.dot(-(LU), x) + b)    #jacobi
        x = np.dot( LD_inv ,(b - np.dot(U, x) ) )  #gauss seidel
        print('paso:', i, '- x:', x)
        if np.linalg.norm(x - xTemp) < eps:    #norm nos da la norma de un vector
            return x   #si converge regresa el valor calculado

    return x    #si no llego a convergencia regresa x
```

Gradiente conjugado

```
In [4]: def conjgrad(A, b, x, tol=1e-8):
            """
            A function to solve [A]{x} = {b} linear equation system with the
            conjugate gradient method.
            More at: http://en.wikipedia.org/wiki/Conjugate_gradient_method
            ========== Parameters ==========
            A : matrix
                A real symmetric positive definite matrix.
            b : vector
                The right hand side (RHS) vector of the system.
            x : vector
                The starting guess for the solution.
            """
            r = b - np.dot(A, x)
            p = r
            rsold = np.dot(np.transpose(r), r)

            for i in range(len(b)):
                print("Iter = ",i)


                Ap = np.dot(A, p)
                alpha = rsold / np.dot(np.transpose(p), Ap)
                x = x + np.dot(alpha, p)
                r = r - np.dot(alpha, Ap)
                rsnew = np.dot(np.transpose(r), r)
                if np.sqrt(rsnew) < tol:
                    break
                p = r + (rsnew/rsold)*p
                rsold = rsnew
            return x
```

```
In [5]: #n = eval(input("Number of equations ==> "))
        #x = np.zeros(n)

        # ---------------------------------------
        import numpy as np
        from scipy import sparse

        X = 10   #valor de prueba
        X = np.random.uniform(size=(6, 6))
        #print(X)

        #X[X < 0.7] = 0   #matriz dispersa
        A = np.array(X)
        b = np.random.uniform(n)
        #x0 = np.random.rand(n)

        b = np.array(n)
        x = np.zeros(n)

        # --------------------------------------

        d = conjgrad(A,  b, x)
```

Conclusiones

Los métodos iterativos permiten la solución de grandes sistemas de ecuaciones que por medio de métodos directos seria sumamente laborioso con una gran cantidad de operaciones

Los lenguajes de alto nivel ofrecen soluciones que permiten realizar la implementación de estos métodos de forma eficiente y con una gran cantidad de bibliotecas disponibles actualmente para realizar operaciones matemáticas.

El uso del lenguaje Julia permite una implementación orientada a las aplicaciones científicas y uso de bibliotecas especializadas con un mejor rendimiento y código organizado.

Referencias web

- https://johnfoster.pge.utexas.edu/numerical-methods-book/LinearAlgebra_IterativeSolvers.html
- https://pythonnumericalmethods.berkeley.edu/notebooks/chapter14.04-Solutions-to-Systems-of-Linear-Equations.html
- https://docs.julialang.org/en/v1/stdlib/Random/
- https://docs.julialang.org/en/v1/stdlib/Random/
- https://www.geeksforgeeks.org/convert-python-list-to-numpy-arrays/

- https://fossies.org/linux/julia/stdlib/SparseArrays/docs/src/index.md
- https://stackoverflow.com/questions/29032946/how-to-efficiently-create-a-sparse-vector-in-python
- https://scipy.github.io/devdocs/search.html?q=sparse+vector
- https://numpy.org/devdocs/user/absolute_beginners.html
- https://www.geeksforgeeks.org/how-to-create-a-vector-in-python-using-numpy/
- https://stackoverflow.com/questions/29032946/how-to-efficiently-create-a-sparse-vector-in-python
- https://docs.julialang.org/en/v1/stdlib/Random/
- https://stackoverflow.com/questions/68423583/how-to-efficiently-construct-a-large-sparsearray-packages-for-this
- https://docs.julialang.org/en/v1/stdlib/Random/
- https://stackoverflow.com/questions/61616906/how-to-search-and-manipulate-sparse-matrices-in-julia

- https://juliahub.com/ui/Packages/ConjugateGradientMethod/y09e6/0.1.0?page=0
- https://stackoverflow.com/questions/46658381/how-to-measure-time-of-julia-program
- https://www.statology.org/python-numpy-linalg-singular-matrix/#:~:text=This%20error%20occurs%20when%20you,zero%20and%20cannot%20be%20inverted.

- MATRICES DISPERSAS Y DETERMINANTES
- https://www.kdnuggets.com/2020/05/sparse-matrix-representation-python.html
- https://stackoverflow.com/questions/2540059/scipy-sparse-arrays
- https://en.wikipedia.org/wiki/Sparse_matrix
- https://en.wikipedia.org/wiki/Determinant