Obliczenia Naukowe- Lista 5

Szymon Skoczylas

10 stycznia 2021

Cel

Celem tego zadania było zaimplementowanie funkcji która rozwiązuje równanie (gdzie A to macierz współczynników i b to wektor prawych stron):

$$Ax = b$$

metodą eliminacji Gauss i rozkładem LU (w dwóch wariantach: z wyborem elementu głównego oraz bez). Zgodnie ze specyfikacją z zadania, wiadomo, że macierz A jest rozmiaru nxn i składa się z podmacierzy o rozmiarach lxl (gdzie l jest podzielne przez n). Każda z podmacierzy jest w postaci przedstawionej w treści zadania. Do przechowywania macierzy została użyta struktura $SparseArrays$ wbudowana w język $Julia$. Zapewnia ona dostęp do elementów niezerowych w czasie kwadratowym, a nie stałym jak przyjmujemy w treści zadania.

Opis i wykonanie

Algorytm eliminacji Gaussa

Algorytm eliminacji Gaussa opiera się na przekształceniu układu równań do macierzy schodkowej górnej. Jest to osiągane tylko za pomocą elementarnych operacji (dodawanie, odejmowanie, mnożenie) na wierszach i kolumanch macierzy (wiersze i kolumny można dowolnie przestawaić). Wynikiem tych operacji jest macierz, na podstawie której (i wektora prawych stron) możemy obliczyć wektor rozwiązań układu równań. Główną operacją algorytmu jest przekształcenie macierzy na trójkątną. W tym celu musimy wyeliminować elementy niezerowe znajdujące się pod przekątną macierzy. Używa się do tego mnożników tj. l_{ij}. Najpierw usuwana jest niewiadoma x_1, z $n-1$ równań, poprzez odejmowanie odpowiednią wielokrotność pierwszego równania od i-tego równania (dla $i = 2, \ldots, n$), aby wyzerować współczynnik x_1. Jest to poprostu wyznaczeniu x_1 z pierwszego równania i podstawieniu do pozostałych z układu. Aby wyzerować element a_{ik} musimy od i-tego wiersza odjąć k-ty wiersz pomnożony przez $l_{ik} = \frac{a_{ik}}{a_{kk}}$. Taki algorytm ma jedną wadę- nie zadziała w przypadku gdy na przekątnej macierzy przy k-tym kroku wystąpi 0. Jedynym sposobem na zniwelowanie tego problemu jest przestawienie kolumn lub wierszy. Aby przystosować ten algorytm do specyficznych postaci macierzy przedstawioncyh w treści zadania, musimy zauważyć, że przy postaci macierzy A, aby otrzymać macierz trójkątną trzeba wyeliminować kolejno następującą liczbę składników: $3, 2, 1, 4, 3, 2, 1, 4, \ldots$. Dla każdej podmacierzy A_k eliminacja obejmuje odpowiednio $l-1, l-2, l-3$ współczynników wa następnie całą ostatnią kolumnę B_k. Każdy wyraz ciągu można określić wyrażeniem $l - k \mod l$ (gdzie k- numer kolumny w macierzy A). Wyjściowy algorytm ma złożność $O(n^3)$, ale przez nasze modyfikacje złożonośc ta przedstawia się $O(nl^3)$, więc jest liniowa. Algorytm jest realizowany przez poniższy kod:

```
function gauss(A, b, n, l)
    vect = Array{Float64}(undef, n)
    for k in 1 : Int64(n/l)
        row_begin = ((k - 1)*l + 1)
        row_fin, row_a_fin, col_fin = min(row_begin + 2*l - 1, n) , (row_begin + l - 1), min(k*l + l, n)
        for i in row_begin : row_a_fin
            for j in i + 1 : row_fin
                sub_coff = A[j, i] / A[i, i]
                A[j, i] = 0
                for m in i + 1 : col_fin
                    A[j, m] = A[j, m] - sub_coff * A[i, m]
                end
                b[j] = b[j] - sub_coff * b[i]
            end
        end
    end
    for k in Int64(n/l):-1:1
        col_fin, row_begin = min(k*l + l, n), ((k - 1)*l + 1)
        row_a_fin = (row_begin + l - 1)
        for i in row_a_fin :-1: row_begin
            sum = 0
            for j in i+1 : col_fin
                sum += A[i, j] * vect[j]
            end
            vect[i] = (b[i] - sum) / A[i, i]
        end
    end
    return vect
end
```

Algorytm Gaussa z wyborem elementu głównego jest modyfikacją standardowego algorytmu Gaussa. Polega ona na wyborze elementu gównego w kolumnie i przestawnieniu wierszy w odpowiedni sposób tak aby dany element znalazł się na przekątnej. Ma to zapobiegać występowaniu zer na przekątnej. Element gółwny jest zawsze największym, z dokładnością co do wartości bezwzględnej, elementem w kolumnie. Przestawianie kolumn można formalnie przedstawić jako:

$$|a_{kk} = |a_{s(k),k}| = max|a_{ik} : i = k, \ldots, n|$$

dla $s(k)$ będącego wektorem permutacji. Algorytm ten jest przystosowany do warunków zadania bardzo podobnie jak normalny algorytm Gaussa, z tą różnicą że dochodzi tutaj jeszcze wybór elementu głównego. Złożoność obliczeniowa tego algorytmu również zostanie taka sama, z dokładnością do stałej (to przez operacje które przypadają na wybór elementu głównego). Algorytm przedstawia się następująco:

```julia
function pivoted_gauss(A, b, n, l)
    p = collect(1:n)
    vect = Array{Float64}(undef, n)
    for k in 1 : Int64(n/l)
        row_begin = ((k - 1)*l + 1)
        row_fin, row_a_fin, col_fin = min(row_begin + 2*l - 1, n), (row_begin + l - 1), min(k*l + l, n)
        for i in row_begin : row_a_fin
            for j in i + 1 : row_fin
                max_r, max = i, abs(A[p[i], i])
                for x in i : row_fin
                    if (abs(A[p[x], i]) > max)
                        max_r, max = x, abs(A[p[x], i])
                    end
                end
                p[i], p[max_r] = p[max_r], p[i]
                sub_cof = A[p[j], i] / A[p[i], i]
                A[p[j], i] = 0
                for m in i + 1 : col_fin
                    A[p[j], m] = A[p[j], m] - sub_cof * A[p[i], m]
                end
                b[p[j]] = b[p[j]] - sub_cof * b[p[i]]
            end
        end
    end
    for k in Int64(n/l):-1:1
        row_begin = ((k - 1)*l + 1)
        col_fin, row_a_fin = min(k*l + l, n), (row_begin + l - 1)
        for i in row_a_fin :-1: row_begin
            sum = 0
            for j in i+1 : col_fin
                sum += A[p[i], j] * vect[j]
            end
            vect[i] = (b[p[i]] - sum) / A[p[i], i]
        end
    end
    return vect
end
```

Rozkład LU

Algorytm rozkałdu LU, jest algorytmem bliskim algorytmowi eliminacji Gaussa, opiera on się na fakcie że wyjściową macierz A można przedstawić jako iloczyn dwóch innych. W algorytmie LU wyznaczane są dwie macierze trójkątne: górną (U) i dolną (L). Otrzymujemy układ:

$$LUx = b$$

którego rozwiązanie sprowadza się do rozwiązania następujących układów

$$Ux = b'$$

$$Lb' = b$$

Aby przystosować tą metode dla warunków wyspecyfikowanych w zadaniu, zmodyfikowane zostały przedstawione wcześniej algorytmy. W taki sposób aby w miejsce usuwanego elementu nie jest podstawiana wartość 0, tylko wartość mnożnika użytego do eliminacji elementu. Wektory x nie są wyliczane oraz aktualizowany wektor b. Dokonywanie tych operacji odbywa się dopiero przy rozwiązywanieu układu równań przy użyciu rozkładu LU. Złożoność tego algorytmu jest taka sama jak dla danego wariantu algorytmu eliminacji Gaussa. Dla funkcji *lu_pivoted* jest dodatkowo podawany wektor perutacji. Algorytmy są przedstawione poniżej (*lu_*- bez wyboru el. głównego, *lu_pivoted*- z wyborem):

```
function lu_(A, b, n, l)
    x = Array{Float64}(undef, n)
    for k in 1 : Int64(n/l)
        row_begin = (k - 1)*l + 1
        row_fin, row_a_fin = min(row_begin + 2*l - 1, n), (row_begin + l - 1)
        for i in row_begin : row_a_fin
            for j in i + 1 : row_fin
                b[j] = b[j] - A[j,i]*b[i]
            end
        end
    end
    for k in Int64(n/l):-1:1
        row_begin = (k - 1)*l + 1
        row_fin, col_fin = (row_begin + l - 1), min(k*l + l, n)
        for i in row_fin :-1: row_begin
            sum = 0
            for j in i+1 : col_fin
                sum += A[i, j] * x[j]
            end
            x[i] = (b[i] - sum) / A[i, i]
        end
    end
    return x
end
```

```
function lu_pivoted(A, b, n, l, p)
    x = Array{Float64}(undef, n)
    for k in 1 : Int64(n/l)
        row_begin = (k - 1)*l + 1
        row_fin, row_a_fin = min(row_begin + 2*l - 1, n), (row_begin + l - 1)
        for i in row_begin : row_a_fin
            for j in i + 1 : row_fin
                b[p[j]] = b[p[j]] - A[p[j], i] * b[p[i]]
            end
        end
    end
    for k in Int64(n/l):-1:1
        row_begin = (k - 1)*l + 1
        row_fin, col_fin = (row_begin + l - 1), min(k*l + l, n)
        for i in row_fin :-1: row_begin
            sum = 0
            for j in i+1 : col_fin
                sum += A[p[i], j] * x[j]
            end
            x[i] = (b[p[i]] - sum) / A[p[i], i]
        end
    end
    return x
end
```

Wyniki

Testy wydajności zostały przeprowadzone dla każdej z zaimplementowanej metod 10-krotnie. Przedstawione dane są uśrednionymi wartościami z tych testów. Do testów użyto danych generowanych przez funkcję *blockmat* z modułu *matrixgen.jl* udostępnionego przez wykładowcę. Uśrednione wyniki testów czasu wykonania (podawane kolejno dla macierzy o rozmiarach 16x16, 10kx10k, 50kx50k):

1. Dla metody eliminacji Gaussa:
    (a) 0.148634s
    (b) 0.187506s
    (c) 4.600984s
2. Dla metody eliminacji Gaussa z wyborem elementu głównego:
    (a) 0.000171s
    (b) 0.207674s
    (c) 4.090996s
3. Dla rozkładu LU:
    (a) 0.000154s
    (b) 0.258636s
    (c) 4.849358s
4. Dla rozkładu LU z wyborem elementu głównego:
    (a) 0.000160s
    (b) 0.196867s
    (c) 4.263090s

Uśrednione wyniki testów zużycia pamięci (podawane kolejno dla macierzy o rozmiarach 16x16, 10kx10k, 50kx50k):

1. Dla metody eliminacji Gaussa:
    (a) 31.969KiB
    (b) 19.312MiB
    (c) 92.206MiB
2. Dla metody eliminacji Gaussa z wyborem elementu głównego:
    (a) 31.763KiB
    (b) 19.386 MiB
    (c) 92.588MiB
3. Dla rozkładu LU:
    (a) 31.449KiB
    (b) 19.312MiB

   (c) 92.206MiB

4. Dla rozkładu LU z wyborem elementu głównego:
   (a) 31.333KiB
   (b) 19.388MiB
   (c) 92.590MiB

Wnioski

Analizując wyniki testów, można powiedzieć, że różnice w czasach działania metod i ich zużycia pamięci są marginalne (np. wyższe wyniki metod z częsciowym wyborem, mogą być efektem dodatkowych operacji składających się na wybór elementu głównego). Widać również, że zużycie pamięci rośnie prawie liniowo w zależności od wartości n. Po wynikach czasowych operacji można wywnioskować, że czas dostępu do elementów macierzy nie jest stały jak podano w treści zadania. Projekt ten pokazuje, że przez modyfikacje klasycznych algorytmów możemy usprawnić rozwiązanie pewnych skomplikowanych problemów.