Scratch

May 6, 2016

In [ ]:
```
#REPORT

AMATH483
Xiao Wang
NetID: Wang19

1.What is the computational complexity of  solve_lower_triangular  and  solve_upper_triangular

The computational complexity of solve_lower_triangular and solve_upper_triangular
is O(n^2).
proof:  In general if both i and j are from 0 to N, it's easy to see O(N^2), FOR example :
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
but in this case, we only run half triangular like this:
0
0 0
0 0 0
0 0 0 0
 This comes out to be 1/2 of N^2 which is still O(N^2).

2.The code I used for 'solve_upper_triangular' is:
```

In [1]:
```
void solve_upper_triangular(double* out, double* U, double* b, int N)
{
  out[N-1] = b[N-1]/U[N*N -1];
  for (int i=(N-2); i>=0; i--){
      out[i] = b[i];
      for (int j=i+1; j<N; j++)
      {
          out[i] -= U[i*N + j]*out[j];
      }
      out[i] = out[i] / U[i*N + i];
  }
}


  File "<ipython-input-1-19847403ec7a>", line 1
2.The code I used for 'solve_upper_triangular' is:
   ^
SyntaxError: invalid syntax
```

a)Does this function have a contiguous memory access pattern? Yes it does have a contiguous memory pattern. By the definition of contiguous memory allocation, contiguous memory allocation is a classical memory allocation model that assigns a process consecutive memory blocks. on the other hand, we can say one without any gaps in the addresses it occupies.We can probably just think of this as a block, and think of something with a gap in the middle as two blocks. In this function outer loop, since i always increases by 1(out[i]=b[i]) ,there is no gaps in the addresses.

b) Assuming the cache create a copy of memory at and after a requested / particular location do you think solve_upper_triangular has any inefficiencies? Why or why not? yes, it does have inefficiencies.For example, when we compute the (N-1)th row, cpmouter will assign a cache to it but we did not really use it and instead, we jump to another row (out[i] -= U[i*N + j]*out[j]).This is also a dis contiguous pattern since U[i*N+j] jump N elements every iteration.

In [ ]:
```
void decompose(double* D, double* L, double* U, double* A, int N)
{
    for (int i=0;i<N;++i)
    {
        D[i*N+i] = A[i*N +i];
    }
    //get lower triangular
    for (int k=1;k < N;++k)
    {
        for (int v=0;v < k;++v)
        {
            L[k*N+v] = A[k*N + v];
        }
    }
    for (int c = N-2; c>=0; --c)
    {
        for (int w = N-1; w>c;--w)
        {
            U[c*N + w] = A[c*N + w];
        }
    }
}
void copy(double* xk,double* out,int N)
{
    for (int i=0; i<N;++i){
        xk[i] = out[i];
    }
}
void gauss_step(double* out, double* S,double* K, double* P, int N, double* xk, double* b, doub
{
    mat_vec(K,L,xk,N,N);//dot(L,xk)
    vec_sub(P,b,K,N);//b-dot(L,xk)
    solve_upper_triangular(out,S,P,N);
}


int gauss_seidel(double* out, double* A, double* b, int N, double epsilon)
{
  double* D = (double*) malloc(N*N * sizeof(double));
  double* L = (double*) malloc(N*N * sizeof(double));
  double* U = (double*) malloc(N*N * sizeof(double));
  double* S = (double*) malloc(N*N * sizeof(double));//(D+U)
```

```
    double* error = (double*) malloc(N * sizeof(double));//out-xk
    double* xk = (double*) malloc(N * sizeof(double);
    double* K = (double*) malloc(N * sizeof(double));//dot(L,xk)
    double* P = (double*) malloc(N * sizeof(double));//b-dot(L,xk)
    decompose(D, L, U, A, N);// call decompose method
    mat_add(S,D,U,N,N);//get the S
    int itr = 1;
    gauss_step(out,S,K,P,N,xk,b,L);
    vec_sub(error,out,xk,N);

     while (vec_norm(error,N) > epsilon)
      {

        K = (double*) malloc(N * sizeof(double));//dot(U,xk)
        P = (double*) malloc(N * sizeof(double));//b-dot(U,xk)
        copy(xk,out,N);//xk=xk1
        gauss_step(out,S,K,P,N,xk,b,L);
        vec_sub(error,out,xk,N);
        ++itr;
    }

    free(D);
    free(L);
    free(U);
    free(xk);
    free(error);
    free(S);
    free(K);
    free(P);
    return itr;
}
```

a)What are the memory requirements as a function of n , the system size? You do not need t o calculate the number of bytes but at least give a description like "n-squared doubles for storing the matrix, n doubles for storing the right-hand side, etc." There are 4 n squared doubles for storing matrices such as D,L,U and S. And 4 n doubles for storing vectors such as error, xk, K and P.

b)What parts of your code have contiguous access patterns? The solve_upper_triangular part which I mentioned previous. And the beginning part of Gauss_seidel and while loop have contigious memory patterns because I preallocate a block of memory large enough to hold the matrix at its final size before entering the loop and C code make computer reserves sufficient contiguous space for the entire full-size array at the beginning of the computation. Once compter has those spaces, we can add elements to the array without having to continually allocate new space for it in memory. What parts do not have contiguous access patterns? Any thoughts on how to improve this, if possible? c) In the while loop, we have to allocate new spaces for K and P which compute the dot product of xk and upper triangular for every iteration to update xk and xk1 and this implementation leads discontiguous. One thing we can improve is that we are given these parameters as arguments at the beginning so we don't have to allocate them every iteration just like this:

int gauss_seidel(double* out, double* K,double* P,double* A, double* b, int N, double epsilon)