数値計算演習
9.11.05
--数値計算(メモリー)--

課題
- 【課題1(必須)】以下の記述をよく読み，2000行2000列の行列を作るプログラムを作れ．wcの結果を提出せよ．
- 【課題2(必須)】4行4列係数行列を持つ連立方程式を解け．MapleによるfsolveあるいはMatrixInverseの結果と比較せよ．連立方程式の解法（逆行列）については，
http://ist.ksc.kwansei.ac.jp/~nishitani/Lectures/NumRecipe/C4.pdf
を参照せよ．
- 【課題3(来週の予定)】連立方程式の解の計算に要する時間を計測せよ．n=2000まで適当に間引いて計測し，そのn依存性をgnuplotで表示せよ．前に作ったshellscriptのautocalcを利用すると便利．
- 【課題4(来週の予定)】gccに-O3のオプションをつけた場合はどうか？

ランダムな値を持った行列の生成
もっとも単純な行列生成プログラム
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define N 10

int main(void){
  int i,j;
  int n;
  double a[N][N];

  scanf("%d",&n);
  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      a[i][j]= 2*(double) random() / RAND_MAX - 1.0;
      printf("%10.5f",a[i][j]);
    }
    printf("\n");
  }
  return 0;
}
```

- 次にこの行列生成プログラムで
```
#define N 2000
```
としてもっと大きな行列を作ってみる．これをコンパイルして，
./a.out > matrix
としてみると
segmentation fault
となることがある(グララボのではならない)．
- 例えば，2000x2000の倍精度の行列に必要なメモリーは
倍精度の浮動小数点数が64bit=8 byteで
2000 x 2000 x8 byte = 32 M byte 程度．
- これはCの実装の問題．これを回避する方法を考える．
- 搭載メモリーの確認は
dmesg | grep Memory
とすればよい．

2次元配列の1次元化
ちょっと改良した次のプログラムを見よ．
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define N 10

void printMatrix(double *a, int n);

int main(void){
  int i,j;
  int n;
  double a[N*N];

  scanf("%d",&n);
  printf("%d¥n",n);

  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      a[i*N+j]= 2*(double) random() / RAND_MAX - 1.0;
    }
  }
  printMatrix(a,n);
  return 0;
}
```

```
void printMatrix(double *a, int n){
  int i,j;

  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      printf("%10.5f",a[i*N+j]);
    }
    printf("\n");
  }
  return;
}
```

- こうすると，2次元の配列を1次元でいじれる．

▼ malloc

▼ メモリーを実行時に動的に確保するmalloc関数を使えば，上記の制限が回避できる．実装は以下の通り．ここでは，連立方程式のためのb行列も作っている．

```
#include <stdio.h>
#include <stdlib.h>

void printMatrix(double *a, double *b, int n);

int main(void){
  int i,j;
  int n;
  double *a;

  scanf("%d",&n);
  printf("%d¥n",n);

  a=(double *)malloc(n*n*sizeof(double));
  b=(double *)malloc(n*sizeof(double));

  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      a[i*n+j]= 2*(double) random() / RAND_MAX - 1.0;
    }
  }

  for (i=0;i<n;i++){
    b[i]= 2*(double) random() / RAND_MAX - 1.0;
  }

  printMatrix(a,b,n);
//  MatrixInverse(a,b,n);
  free(a);
  free(b);
  return 0;
}

void printMatrix(double *a, double *b, int n){
  int i,j;

  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      printf("%10.5f",a[i*n+j]);
    }
    printf(":%10.5f",b[i]);
    printf("¥n");
  }
  printf("¥n");
  return;
}
```

```
[BobsNewPBG4-2:~/NumRecipeEx05/Matrix] bob% ./a.out > matrix
2000
[BobsNewPBG4-2:~/NumRecipeEx05/Matrix] bob% wc matrix
    2001 4000001 40002005 matrix
```

▼ 逆行列の計算

▼ 以下の逆行列をガウスの消去法で求めるサブルーチンを組み込んで，逆行列を求める．fabsを使っているので，math.hのincludeを忘れないように．

```
int MatrixInverse(double *a, double *b, int n){
    double *x;
    double pvt=0.00005,am;
    int i,j,k;
```

```
    x=(double *)malloc(n*sizeof(double));

    for(i=0;i<n-1;i++){
      if(fabs(a[i*n+i])<pvt){
        printf("Pivot %3d=%10.5f is too small.¥n",i,a[i*n+i]);
        return 1;
      }
      for(j=i+1;j<n;j++){
        am=a[j*n+i]/a[i*n+i];
        for(k=0;k<n;k++) a[j*n+k]-=am*a[i*n+k];
        b[j]-=am*b[i];
      }
    printMatrix(a,b,n); //大きな計算の時には消す.
    }
    //Backward substitution
    for(j=n-1;j>=0;j--){
      x[j]=b[j];
      for(k=j+1;k<n;k++){
        x[j]-=a[j*n+k]*x[k];
      }
      x[j]/=a[j*n+j];
    }
    printMatrix(a,x,n);
    free(x);
    return 0;
  }
```

▼ 計算時間の計測

▼ 実行時間を計る方法.

```
#include <stdio.h>
#include <time.h>

int main(void){

// time関数を利用
  time_t ts,te;
  double t;

  time(&ts);
  printf("%s\n",ctime(&ts));
  printf("Hello\n");
  time(&te);
  t=difftime(te,ts);
  printf("%4.1f\n", t);

// clock関数を利用
  clock_t start, end;

  start = clock();
  printf("Hello\n");
  end = clock();
  printf("%5.2f\n", (double)(end-start)/CLOCKS_PER_SEC);
  return 0;
}
```

- ここでは，clock関数を使う．他にgetrusage,gettimeofdayなどを利用して計測されている.
-