1　構造体配列　C 言語　160 頁

1.1　学習のポイント

構造体データをいくつかまとめた構造体配列について学びます。

先の木星の 4 大衛星のデータは 4 件の構造体データから成っていますから、このようなデータは構造体を用いて次のように扱うと便利です。

```
static struct eisei a[] = {{'I', 5, 1.7691},
                           {'E', 6, 3.5512}
                           {'G', 5, 7.1545}
                           {'C', 6, 16.6890}};
```

このようなデータにおいてイオのデータを参照するには、

```
a[0].name
a[0].kodo
a[0].shuki
```

とします。
「C 言語」(河西朝雄著　ナツメ社)160 頁

1.2　例題 37 reidai37.c

練習問題 37　衛星名を'I'、'E' などの文字型ではなく、"Io", "Europa"などの文字列で表せるように例題 37 を改良しなさい。

```
/*
  木星の 4 大衛星のデータを構造体配列に初期化し、そのデータをディスプレイに
  表示しなさい。
  例題 37
  reidai37.c
*/

#include <stdio.h>

  struct eisei{
```

```
    char name;
    int  kodo;
    double shuki;
  };

int main()
{
  int i;
  static struct eisei a[] = {{'I', 5, 1.7691},
                             {'E', 6, 3.5512},
                             {'G', 5, 7.1545},
                             {'C', 6, 16.6890}};

  for(i = 0; i < 4; i++){
    printf("%c%3d%8.4f\n", a[i].name, a[i].kodo, a[i].shuki);
  }

  return 0;
}
```

2　AI による大規模データ処理入門

進化計算は、私たちの暮らしのさまざまな場面で活用されています。

たとえば、新幹線のモデル N700 系の先頭車両設計では、独特の形 (フォルム) を作るうえで大きな役割を果たしています。

N700 系は、従来よりも 20km 速い速度 270km でカーブを曲がれる性能を持ちます。しかし、それまでの先頭車両の形状では、スピードアップした分、騒音の原因となるトンネル微気圧波が拡大してしまいます。

「エアロ・ダブルウィング」と呼ばれる独特のフォルムは、進化計算を用いたおよそ 5000 回のシュミレーションで導き出された最適値です。

また、国産のジェット機の翼の設計においては、多目的進化計算と呼ばれる手法が用いられました。

この手法により、ジェット旅客機の燃費効率の向上と機外騒音の低減という二つの目標を同時に最適化し、競合機よりも性能を改善することに成功しています。

進化計算と深層学習　15 頁　伊庭斉志　オーム社

2.1 群知能アルゴリズム

生物の群れの挙動を模倣することで大規模データを処理する群知能 (swarm intelligence) と呼ばれる手法について説明します。
群知能では、それぞれが独立して動作する固体が複数集まり、ある一定の規則のもとで行動します。この際に、個体同士が影響し合ったり、環境との相互作用を行なうことで、固体の状態を変化させます。このような動作を繰り返すことで、固体の状態変化を通して問題の解を得るのが群知能アルゴリズムです。

群知能による問題解決アルゴリズムとして、粒子群最適化法 (Particle Swarm Optimization) を取り上げます。

2.2 粒子群最適化法 pso.c

```
/************************************/
/*     粒子群最適化法プログラム       */
/*            pso.c                 */
/*                                  */
/************************************/

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

/*    記号定数                       */
#define NOPS 10 /*粒子の個数*/
#define LIMITL 256 /*配列最大領域*/
#define ILIMIT 100 /*繰り返しの回数*/
#define SEED 32767 /*乱数の初期値*/
#define W 0.7 /*慣性定数*/
#define C1 1.4 /*ローカルな質量*/
#define C2 1.4 /*グローバルな質量*/

/*  平面座標を表現する構造体         */
struct point{
 double x ;
 double y ;
} ;
```

```
/*      粒子を表現する構造体          */
struct particle{
 struct point pos ;/*位置*/
 double value ;/*評価値*/
 struct point v ;/*速度*/
 struct point bestpos ;/*最適位置*/
 double bestval ;/*最適評価値*/
} ;


/*    関数のプロトタイプの宣言        */
void initps(struct particle ps[]) ;
                    /*粒子群の初期化*/
void printps(struct particle ps[]) ;
                     /*粒子群の表示*/
double frand() ;/*実数乱数*/
double calcval(double x,double y) ;/*評価値の計算*/
void optimize(struct particle ps[]) ;
                   /*粒子群の位置更新*/
void setgbest(struct particle ps[]) ;
          /*  群れの最適位置を格納   */

/*大域変数*/
struct point gbestpos ;/*群中の最適位置*/
double gbestval ;/*群中の最適評価値*/


/**********************/
/*     main() 関数      */
/**********************/
int main()
{
 struct particle ps[LIMITL] ;/*粒子群*/
 int i;/*繰り返し回数の制御*/

 /*乱数の初期化*/
 srand(SEED) ;

 /*粒子群の初期化*/
 initps(ps) ;
```

```
 printps(ps) ;

 /*最適化の本体*/
 for(i=0;i<ILIMIT;++i){
  optimize(ps) ;
  printf("%d 回目\n",i) ;
  printps(ps) ;
 }

 return 0 ;
}

/*************************/
/*    optimize() 関数      */
/*    粒子群の位置更新     */
/*************************/
void optimize(struct particle ps[])
{
 int i ;
 double r1,r2 ;/*乱数の値を格納*/

 for(i=0;i<NOPS;++i){
  /*乱数の設定*/
  r1=frand() ;
  r2=frand() ;
  /*速度の更新*/
  ps[i].v.x=W*ps[i].v.x
     +C1*r1*(ps[i].bestpos.x-ps[i].pos.x)
     +C2*r2*(gbestpos.x-ps[i].pos.x) ;

  ps[i].v.y=W*ps[i].v.y
     +C1*r1*(ps[i].bestpos.y-ps[i].pos.y)
     +C2*r2*(gbestpos.y-ps[i].pos.y) ;

  /*位置の更新*/
  ps[i].pos.x+=ps[i].v.x ;
  ps[i].pos.y+=ps[i].v.y ;

  /*最適値の更新*/
  ps[i].value=calcval(ps[i].pos.x,ps[i].pos.y) ;
```

```
  if(ps[i].value<ps[i].bestval){
   ps[i].bestval=ps[i].value ;
   ps[i].bestpos=ps[i].pos ;
  }
 }
  /*群中最適値の更新*/
 setgbest(ps) ;
}


/*************************/
/*    calcbal() 関数       */
/*    評価値の計算         */
/*************************/
double calcval(double x,double y)
{
 return x*x+y*y+1 ;
}

/*************************/
/*    setgbest() 関数      */
/*  群れの最適位置を格納   */
/*************************/
void setgbest(struct particle ps[])
{
 int i ;
 double besti ;
 double x,y ;

 besti=ps[0].value ;
 x=ps[0].pos.x ;
 y=ps[0].pos.y ;

 for(i=0;i<NOPS;++i)
  /*現在の最良評価値を探す*/
  if(ps[i].value<besti){
   besti=ps[i].value ;
   x=ps[i].pos.x ;
   y=ps[i].pos.y ;
  }
```

```
 /*評価値が過去よりもよかったら更新*/
 if(besti<gbestval){
  gbestval=besti ;
  gbestpos.x=x ;
  gbestpos.y=y ;
 }
}

/*************************/
/*    initps() 関数      */
/*    粒子群の初期化     */
/*************************/
void initps(struct particle ps[])
{
 int i ;
 double x,y ;

 for(i=0;i<NOPS;++i){
  /*位置*/
  x=ps[i].pos.x=frand()*2-1.0 ;
  y=ps[i].pos.y=frand()*2-1.0 ;
  /*評価値*/
  ps[i].value=calcval(x,y) ;
  /*速度*/
  ps[i].v.x=frand()*2-1.0 ;
  ps[i].v.y=frand()*2-1.0 ;
  /*最適位置*/
  ps[i].bestpos.x=ps[i].pos.x ;
  ps[i].bestpos.y=ps[i].pos.y ;
  /*最適評価値*/
  ps[i].bestval=ps[i].value ;
 }
 /*群れの最適位置を格納*/
 gbestval=ps[0].value ;
 gbestpos.x=ps[0].pos.x ;
 gbestpos.y=ps[0].pos.y ;
 setgbest(ps) ;
}
```

```
/*************************/
/*   printps() 関数       */
/*    粒子群の表示        */
/*************************/
void printps(struct particle ps[])
{
 int i ;

 for(i=0;i<NOPS;++i){
  printf("%d ",i);
  printf("%lf %lf ",ps[i].pos.x,ps[i].pos.y) ;
  printf("%lf ",ps[i].value) ;
  printf("%lf %lf ",ps[i].v.x,ps[i].v.y) ;
  printf("%lf %lf ",
       ps[i].bestpos.x,ps[i].bestpos.y) ;
  printf("%lf\n",ps[i].bestval) ;
 }
 printf("%lf %lf %lf\n",gbestpos.x,gbestpos.y,gbestval) ;
}

/*************************/
/*   frand() 関数         */
/*   実数乱数            */
/*************************/
double frand(void)
{
 double result ;
 while((result=(double)rand()/RAND_MAX)>=1) ;
 return result ;
}
```

2.3 実行方法

```
./pso
```