粒界の原子間ポテンシャルモデルの構造緩和

関西学院大学理工学部
情報科学科 西谷研究室 6712 松塚景子

平成 23 年 2 月 21 日

概 要

Read-Shockley と大槻氏により小傾角粒界エネルギーを求める式のどちらが正当かを検証のため，粒界のモデルを作成し粒界の最安定構造を探るためにエネルギーの最小値を導く必要がある．本研究は MC(モンテカルロ) 法を用いて関数の最小値を求めるプログラムを作成し，粒界モデルのエネルギーを求めるプログラム中に組み込んだ．また，Conjugate Gradient(CG:共役勾配) 法，Powell 法を用いて関数の最小値を求めるプログラムを作成し，計算速度を検討した．

目 次

第1章　背景

これまで小傾角粒界のエネルギーは Read-Shockley の提唱した式で示されると考えられていた．近年，大槻によりそれとは異なる考えが示されたが，その正当性については未だ証明されていないため検証する必要があり．西谷研究室でも 3 人のメンバーが分担してこの問題に取り組んでいる．この内容は小傾角粒界のエネルギーの解析界を求め，それと数値解とを比較し，大槻の小傾角粒界モデルの妥当性を検討するものである．

数値解を求めるシミュレーションを行う上で，最安定となる小傾角粒界モデルのエネルギーと，その原子位置を求める際に，モデルの構造最適化を行う必要がある．構造緩和には内部緩和と外部緩和がある．図 1.1 は内部緩和の模式図であり，図のように粒界の 1 ブロックの中で原子 1 つ 1 つを動かし，1 ブロック内での最安定構造を探る．一方，図 1.2 は外部緩和の模式図であり図のように粒界のかたまりである 1 ブロックを動かし．最安定構造を探る．最安定構造とは，粒界の構造エネルギーが低い状態のことを指し，本研究では内部緩和に MC(モンテカルロ) 法を，外部緩和に CG(共役勾配) 法または Powell 法を用いてエネルギーの最小値を求める．図 1.3 は外部緩和を行う CG 法と Powell 法のプログラムの流れである．CG 法で frprmn,rb を，Powell 法で Powell.rb を使用する．

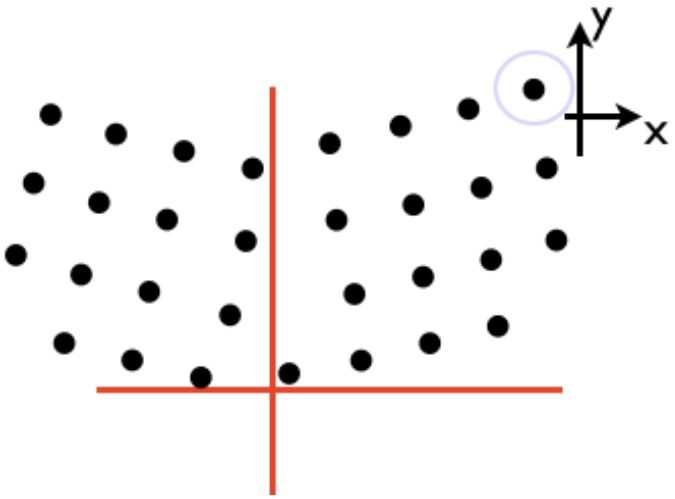

図 1.1: MC 法を用いる内部緩和

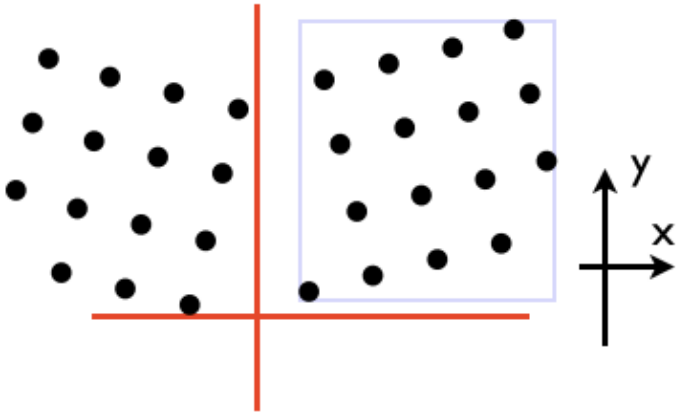

図 1.2: CG 法または Powell 法を用いる外部緩和

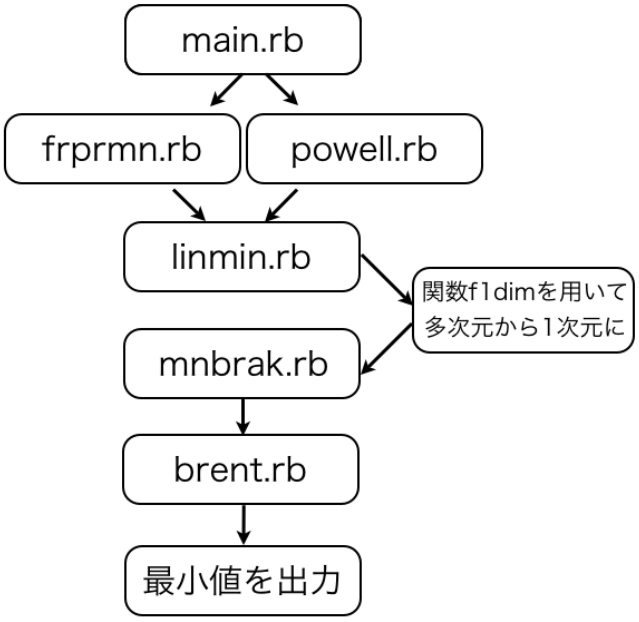

図 1.3: 外部緩和のプログラムのフローチャート

1.1 Maple

Mapleは数式処理，数値計算，グラフ作成などを行うソフトウェアである．Mapleを使用すると微分積分，行列ような様々な数学の計算や作図ができる．本研究ではMC法を用いる巡回セールスマン問題，CG法，Powell法での計算結果が妥当かどうかの判断のためにグラフで視覚化するのに使用した．

1.2 Ruby

Rubyは，まつもとゆきひろ氏により開発されたオブジェクト指向スクリプト言語である．たのしくプログラムを書くということを目的に開発されており，シンプルな文法でプログラムを作成することができる．しかし，計算速度がC言語やfortranに比べると遅いのが欠点である．

第2章　最適化手法

2.1　MC(モンテカルロ) 法

MC 法は．大規模な最適化問題，特に真の最大 (最小) がたくさんの極大 (極小) に隠れている場合に向く方法として，注目を集めている方法である．2.1.1 で述べる，巡回セールスマン問題 (Traveling Salesman Probrem,TSP) や複雑な集積回路の設計にも使用されて成功を収めている．MC 法の応用はいずれも組み合わせ最小化の例である．1953 年に Metropolis たちが初めてこの種の原理を数値計算に取り入れた．このアルゴリズムは熱力学を模した系がつぎつぎにいろんな状態に出会う．元のエネルギーが E_1 で，出会った状態のエネルギーが E_2 なら，確率 $p = \frac{-exp(E_2-E_1)}{kT}$(k は Boltzmann 常数) で新しい状態に遷移する．もし $E_2 < E_1$ ならこの値は 1 より大きくなるが，このときは $p = 1$ とし，必ず新しい状態に遷移する．MC 法は常に坂を登る (下る) 一方ではないため，図 2.1 のような極小がいくつもあるような関数の最小値を求めるのに適している．プログラムは単純だが，新しい状態をランダムに決定するため，無駄いため多く収束が遅くあいまいである．後に記す巡回セールスマン問題 (Traveling Salesman Probrem,TSP) で，新しいルートを採用するかどうかの判断に MC 法が使用されており，プログラムが正常に動作するかどうかの判断に Maple で TSP を解くプログラムを作成した [1, pp.109-115, 2, pp.330-338].

2.1.1　巡回セールスマン問題

巡回セールスマン問題とはセールスマンが 3 つ以上の都市をちょうど一度ずつ巡り出発地に戻る巡回路の総移動距離が小さい順路を求める組み合わせ最適化問題である．最初に設定された順路から新たな順路に変更するとき，順路はランダムに作成される，この時，新たな順路を採用するかどうかの判断に MC 法が使用されている．

2.1.2　内部緩和での MC 法の利用方法

MC 法は結晶の内部緩和で使用される．ランダムに原子を動かして動かす前と動かした後の結晶の構造エネルギーを計算し，動かした後の方がエネルギーが小

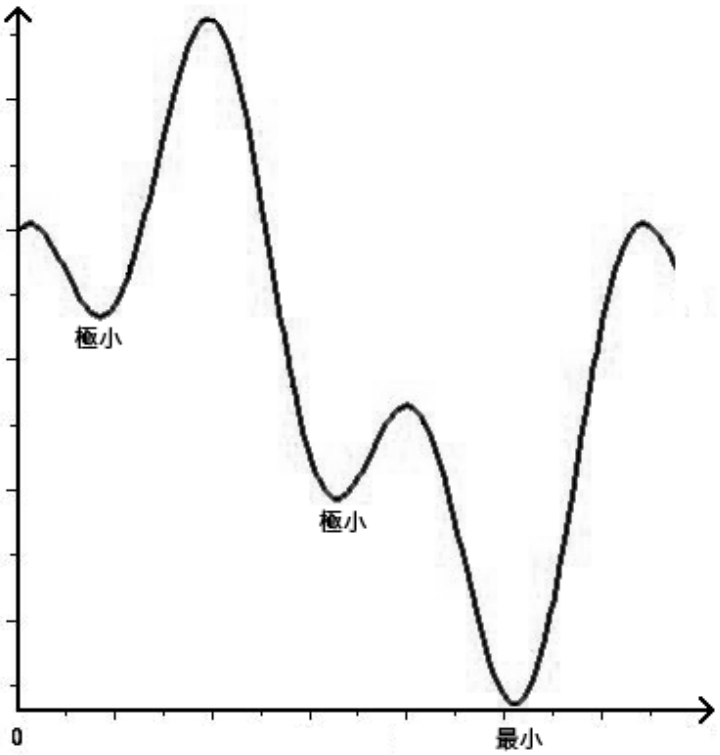

図 2.1: MC 法に適した関数

さければ新しい位置を採用，大きくても $exp(-dE/T)$ の確率で新しい位置を採用する．

2.2 CG(共役勾配) 法

CG 法は多次元空間関数の最小値を求める手法である．N 次元空間の点 $\boldsymbol{P}$ で，関数値 $f(\boldsymbol{P})$ だけなく，勾配 (1 階偏導関数のベクトル)$\nabla f(\boldsymbol{P})$ も計算できる場合，勾配の情報を使うことが有用である．勾配を使うなら，勾配の計算ごとに N 成分の情報が得られる．これをうまく使えば直線に沿った最小化は N 回程度ですむので計算時間が大幅に短縮できる．直線に沿った極小での新しい勾配の向きはもとの直線に垂直になる．したがってこの方法では必ず直角に曲がらなければならず，新しい方向は一般に最小に導かない．CG 法は初期状態から次の状態に進むとき，新しい勾配方向にすすむのでなく何らかの方法で古い勾配方向と共役な方向に進んでいく．CG 法は MC 法と比べ収束がはやい．共役な方向とは勾配 ∇f が $0 = \boldsymbol{u} \cdot \nabla f$ となるような方向 $\boldsymbol{u}$ のことである．二つの最も重要な共役勾配法は Fletcher-Reeves 法と Polak-Ribiere 法であり，どちらの手法を用いるかを　関数 frprmn の中で選択できるようになっている．これらの二つの方法は密接に関連しあっている．$\boldsymbol{A}$ を正定値の $n \times n$ 対称行列 (n は次元数)，$\boldsymbol{g_0}$ を任意のベクトルとし，$\boldsymbol{h_0} = \boldsymbol{g_0}$ とおく．$i = 0, 1, 2...$ について次の 2 系列のベクトルを定義する．

$$\boldsymbol{g_{i+1}} = \boldsymbol{g_i} - \lambda_i \boldsymbol{A} \cdot \boldsymbol{h_i} \qquad \boldsymbol{h_{i+1}} = \boldsymbol{g_{i+1}} + \gamma_i \boldsymbol{h_i} \tag{2.1}$$

．ここで，λ_i，γ_i は $\boldsymbol{g_{i+1}} \cdot \boldsymbol{g_i} = 0$，$\boldsymbol{h_{i+1}} \cdot \boldsymbol{A} \cdot \boldsymbol{h_i} = 0$ を満たすように選ぶ．すなわち

$$\lambda = \frac{\boldsymbol{g_i} \cdot \boldsymbol{g_i}}{\boldsymbol{g_i} \cdot \boldsymbol{A} \cdot \boldsymbol{h_i}} \qquad \gamma_i = -\frac{\boldsymbol{g_{i+1}} \cdot \boldsymbol{A} \cdot \boldsymbol{h_i}}{\boldsymbol{h_i} \cdot \boldsymbol{A} \cdot \boldsymbol{h_i}} \tag{2.2}$$

(ただし分母が 0 なら $\lambda_i = 0$，$\gamma_i = 0$) とする．すると，すべての $i \neq j$ について次式が成り立つ．

$$\boldsymbol{g_i} \cdot \boldsymbol{g_j} = 0 \qquad \boldsymbol{h_i} \cdot \boldsymbol{A} \cdot \boldsymbol{h_j} = 0 \tag{2.3}$$

言い換えれば手続き 2.1 は一種の Gram-Schmidt の双直交化で，各 $\boldsymbol{g_i}$ を直前のもの $\boldsymbol{g_{i-1}}$ と直交のものにし，各 $\boldsymbol{h_i}$ を直前のものと共役にする．実際，各 $\boldsymbol{g}$ は互いに直交で各 $\boldsymbol{h}$ は互いに共役となる，次の λ_i，γ_i の式が 2.2 の式と同値であることが示せる．

$$\gamma_i = \frac{\boldsymbol{g_{i+1}} \cdot \boldsymbol{g_{i+1}}}{\boldsymbol{g_i} \cdot \boldsymbol{g_i}} = \frac{(\boldsymbol{g_{i+1}} - \boldsymbol{g_i}) \cdot \boldsymbol{g_{i+1}}}{\boldsymbol{g_i} \cdot \boldsymbol{g_i}} \tag{2.4}$$

$$\lambda_i = \frac{\boldsymbol{g_i} \cdot \boldsymbol{h_i}}{\boldsymbol{h_i} \cdot \boldsymbol{A} \cdot \boldsymbol{h_i}} \tag{2.5}$$

式 2.1 が成立するなら，Fleycher-Reeves．Polak-Ribiere の方法には 1 カ所だけ小さな違いがあるが，その違いはときには重要になる．Fletcher-Reeves は最初 γ_i についての式 2.4 の第 1 のものを使い，Polak-Riboere は後にこの式の第 2 のものを

使うことを提案した．これらが同値なのは厳密な 2 次形式についてであり，現実の関数は厳密な 2 次形式ではなく，2 次形式と仮定して到達した点からさらに反復を続けなければならない．Polak-Ribiere の公式の方が反復間のつなぎ目がより優雅である．この方法は，うまく行かなくなってくると $\boldsymbol{h}$ をその場所での勾配に再設定する傾向があるが，これは共役勾配の手続きを新しく再開することと同値である．

本研究で作成した CG 法は main.rb で関数 frprmn を呼び出す．frprmn は直線上を最小化を行う linmin を呼び出す．linmin の中で最小値を求めたい関数 func を f1dim を用いて人為的に多次元から 1 次元にし，直線上の最小化を行う．linmin の中では直線の最小化をする関数 Brent と最小化をする時に最初に囲い込む点を決める mnbrak を呼び出している．を Brent 法は 1 次元における最適化手法の 1 つで黄金分割法と放物線補間を組み合わせたものであるので黄金分割法と放物線補間を先に説明する [2, pp.282-310]

2.2.1 黄金分割法

黄金分割法とは黄金比と $\tau = \frac{1+\sqrt{5}}{2}$ とする．図 2.2 のように区間 a，b 内に最小値となる点が 1 つあるとき，a と b の距離を L とする．新たな点 c，d を $c = a + \frac{\tau-1}{\tau}L$, $d = a + \frac{1}{\tau}L$ を求める．つまり c は L を $1-\frac{1}{\tau}$，$\frac{1}{\tau}$ に内分する点，d は L を $\frac{1}{\tau}$，$1-\frac{1}{\tau}$ に内分する点である．$f(c) > f(d)$ なら，$a = c$ とし，$f(c) \leqq f(d)$ なら $b = d$ とする．以上を繰り返し徐々に最小値のある範囲を狭めていき最小値を導く [2, pp.285-287, 3].

2.2.2 放物線補間

放物線補間とは図2.3にあるように，初期点は極小を挟む３つの点①，②，③(①<②<③)をとる．その 3 点を通る曲線の極小での点④とする．次に①，②，④を通る曲線の極小での点を⑤とする．以上を繰り返して最小値を求める [2, pp.289-290].

2.2.3 外部緩和での CG 法の利用方法

作成した CG 法は結晶構造の外部緩和で利用される．利用方法は，frprmn で使用されている関数 dfunc 内で p1 を微分するときに微小な値 dd を足し合わせている．dd が外部緩和では粒界をいくら動かすかに相当する．しかし，粒界でできた一塊の 1 ブロックが必ずしも微分可能とは限らない．CG 法は微分をする必要があるので粒界の外部緩和には適していない可能性もある．そのために CG 法とは別に微分を必要としない最小化を求める手法，修正 Powell 法も作成する必要がある．

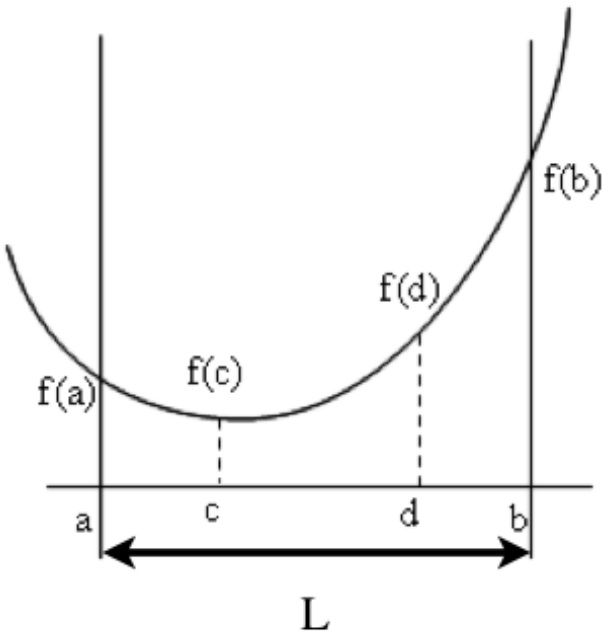

図 2.2: 黄金分割法

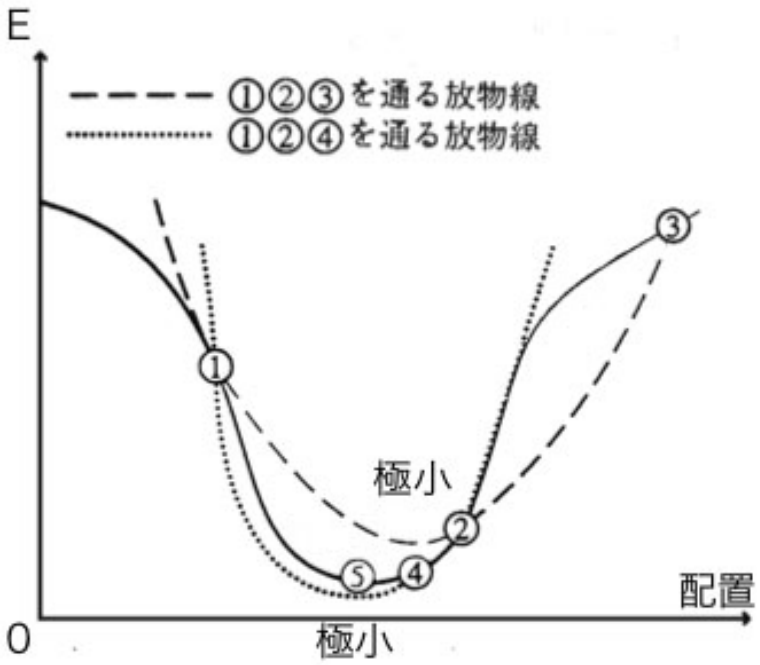

図 2.3: 放物線補間

2.3 修正Powell法

Powell法は勾配を求めることができない関数の最小化に用いられる．Powell法はmain.rbで関数Powellを呼び出し，Powellで直線上の最小化を行うlinminを呼び出す．linminの中でmnbrakとBrentを呼び出して最小値を求めるところはCG法と同じである．Powellが発見したこの方法は，まず，方向集合として単位ベクトル$\boldsymbol{e_1}.\boldsymbol{e_2},\ldots,\boldsymbol{e_N}$(Nは次元数)をとり．$\boldsymbol{u_i} = \boldsymbol{e_i}$とする．出発点を$\boldsymbol{P_0}$とおく．$i = 1,..,N$について$\boldsymbol{P_{i-1}}$を方向$\boldsymbol{u_i}$に沿った最小に移動し，その点$\boldsymbol{P_i}$とおく．$i = 1,..,N$について$\boldsymbol{u_i} = \boldsymbol{u_{i+1}}$とおく．$\boldsymbol{u_N} = \boldsymbol{P_N} - \boldsymbol{P_0}$とおく．$\boldsymbol{P_N}$を方向$\boldsymbol{u_N}$に沿った最小に移動し，その点を$\boldsymbol{P_0}$とおく．同様に繰り返して，方向集合の全要素を一巡することを何度も繰り返して関数値が減少しなくなったら終了する．最小値が求まり繰り返すたびに$\boldsymbol{u_i} = \boldsymbol{u_{i+1}}$とする，しかし，この方法だけでは各段階で新たに方向集合をつくるときに線形従属になる傾向がある．こうなったとき，関数の最小値はN次元空間でなく，部分空間における最小値を関数の最小値として，間違った答えを求める．そのため，欠点を克服した修正Powell(以下，Powell)法は．関数が最も速く減少した方向を捨てることで線形従属になることを避ける[2, pp.282-306].

第3章　結果

3.1　巡回セールスマン問題

3.1.1　Mapleでの巡回セールスマン問題

まず，巡回セールスマン問題で最適解を与えるプログラムをMapleで作成した．これは，Mapleは計算した結果を簡単に図に表すことができるため，最適化する前と，最適化した後の順路を図で表せば，プログラムが正当に作動しているかどうかの判断か容易であるからである．

a,b間の距離を計算する関数

```
Distance:=proc(a,b)
return sqrt((Position[a][1]-Position[b][1])^2
          +(Position[a][2]-Position[b][2])^2);
end proc;
```

ルートpでの総距離を計算する関数

```
E:=proc(p)
local S:=0,i;
 for i from 1 to N_city do
  S:=S+dis[p[i],p[i+1]];
 end do;
 return S;
end proc;
```

a,b の都市の廻る順番を入れ替える関数

```
da:=proc(a,b)
  local t;
  global Path;
  t:=Path[a];
  Path[a]:=Path[b];
  Path[b]:=t;
  return Path;
end proc;
```

※ Maple では配列の最初の要素番号を 0 番目とせず 1 番目とする．

廻る都市数を N_city とする．今回は N_city=10 として，スタート地点を含む 10 都市を廻る．10 都市の (x,y) 座標をランダムに生成し配列 Position に格納していく．

```
Position:=seq([evalf(rand()/10^12),evalf(rand()/10^12)],i=1..N_city);
```

Position[1] は 1 番目 (スタート地点) の (x,y) 座標を表し，Position[1,1]，Position[1,2] はそれぞれ 1 番目の x 座標，y 座標を表す．

Path:=[1,2,..,N_city,1] とする．
Path[N_city+1]=1 である理由は最後の都市を廻った後にスタート地点に戻るという意味である．

配列 Real_Path=[] の i 番目の要素に Position[i] を格納．最後はスタート地点に戻るために Real_Path の N_city+1 番目の要素は Position[1] である．

```
Real_Path:=[seq(Position[Path[i]],i=1..N_city),Position[1]]:
```

これで，n 番目に巡回する都市の座標は Real_Path[n] に格納されたことになる．最適化される前の巡回路，Real_Path をプロットすると図 3.1 のようになり，とても無駄が多い順路であることがわかる．

巡回路を最適化するために，まず，それぞれの都市間の距離を計算し配列 dis[] に格納する．

```
dis:=array(1..N_city,1..N_city);
for i from 1 to N_city do
 for s from 1 to N_city do
  dis[i,s]:=Distance(i,s);
 end do;
end do;
```

最適化される前の状態の総距離 E(Path) は 4.375005263 である．総距離の増減の変化を視覚化するためにルートを変更するたびに配列 E_trace[] に総距離を格納していく．

s1,s2 に 2 から N_city からランダムに選ばれた値を代入し s1,s2 を入れ替えたルートの方が総距離が小さければ入れ替えを採用し新しいルートを作成，大きい場合でも exp(-dE/T) の確率で入れ替えを採用する. 以上を for ループで繰り返し，その都度 E_trace に総距離を保存する．

```
  sel_city:=rand(2..N_city):  #2 から N_city の中から数をランダムに選ぶ
sel_city をつくる
  s1:=sel_city();
  s2:=sel_city():
  dela:=da(s1,s2);
  dE:=E(dela)-E(a);



  if (dE<=0) then
    a:=dela;
    E_trace:=[op(E_trace),E(a)];
  elif (dE>0) then
    pb:=exp(-dE/T);
    rd:=pb_rd()*10^(-5);
    if (rd<pb) then
      a:=dela;
      E_trace:=[op(E_trace),E(a)];
    end if;
  end if;
```

ルート変更後の順路は，プロットすると図 3.2 の通りである．また，E_trace の変化は図 3.3 の通りである．

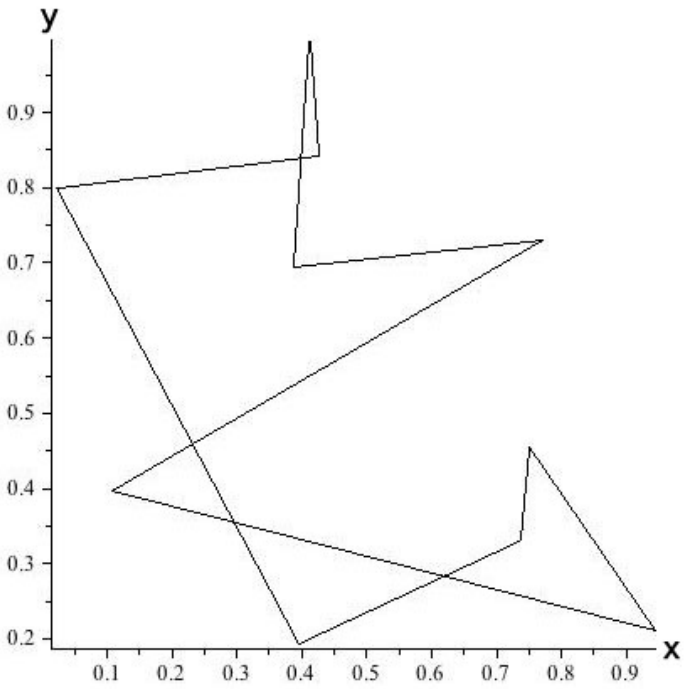

図 3.1: 最適化する前の順路

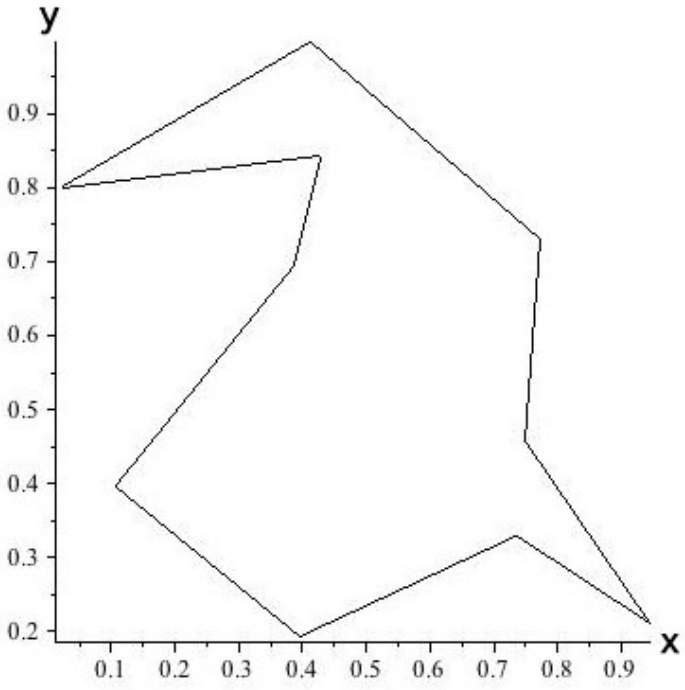

図 3.2: 最適化した後の順路

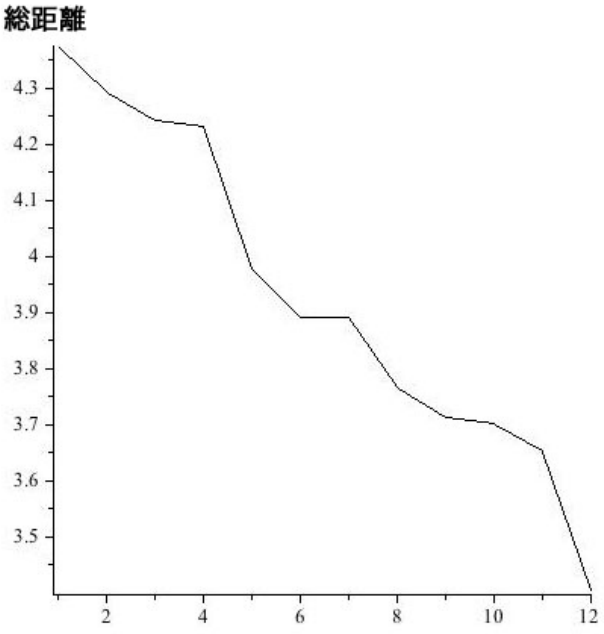

図 3.3: 総距離 (E_trace) の変化

3.1.2 Rubyでの巡回セールスマン問題

Rubyでのプログラムの流れは以下の通りである.

a,b 間の距離を計算する関数

```
def Distance(a,b)
  return sqrt( ((Position[a][0]-Position[b][0])**2)+
                  ((Position[a][1]-Position[b][1])**2))
end
```

ルート a での総距離を計算する関数

```
def E(a)
  s = 0
  for i in 0..N_city-1
      s = s + $dis[[a[i],a[i+1]]]
  end
  return s
end
```

セールスマンがまわる都市数 N の座標 (x,y) をランダムに決定し, スタート地点の都市を 0，最後に行く都市を N-1 とし，n 番目の都市の x 座標を配列 Position_x[n-1]，y 座標を配列 Position_y[n-1] に格納していく．また，配列 Position[n] には Position_x[n],Position_y[n] を格納する，

```
for i in 0..N_city-1
  Position_x[i]=rand()
  Position_y[i]=rand()
  Position << [Position_x[i],Position_y[i]]
end
```

配列 Path[]=[0,1,2,..,N,0] とし，この配列に格納されている数字は都市名であり，セールスマンが巡回する順番である．すべての都市を巡った後スタート地点である 0 に戻るため，Path[N_city]=0 とする.

```
for i in 0..N_city-1
```

```
  Path[i]=i
end
Path[N_city]=0
```

それぞれの都市間の距離を計算し$dis に格納する．

```
for i in 0..N_city-1
  for j in 0..N_city-1
  $dis[[i,j]] = Distance(i,j)
  end
end
```

Path の複製 dela[] を作成する．

```
a = Path
dela = []
a.each do |ai|
  dela << ai
end
```

巡回する都市の中からランダムに 2 都市 s1,s2 を選択し，選んだ 2 都市間で巡回路を変更する．ここで，最初のルート Path[] よりも s1,s2 を入れ替えた方が総距離が小さくなるなら新たなルートを採用，大きくなる場合も exp(-dE/T) の確率で採用する．新たなルートは配列 a[] に格納される．

```
e_trace=[]
300.times do
  s1=rand(N_city-1)+1
  s2=rand(N_city-1)+1

  dela = []
  a.each do |ai| dela << ai end

  f=dela[s1]
  dela[s1]=dela[s2]
  dela[s2]=f

  dE=E(dela)-E(a)

  if dE<0 then
```

```
    a=dela
    e_trace << E(a)
  elsif dE>0 then
    pb=exp(-dE/T)
    rd=rand(10000000)*(10**-7)
    if rd<pb then
      a=dela
      e_trace << E(a)
    end
  end
end
```

3.2 CG法

CG法で使用する関数を以下にまとめる.

etc.rb 内の fmax(a,b)

a,b の値を入力し大きい方の値を返す関数.

```
def fmax(a,b)
  if a>=b then return a
  else return b end
end
```

etc.rb 内の mysign(a,b)

a,b の値を入力し，b が 0 以上なら c=|a| を b が 0 未満なら c=-|a| を返す関数. a に b と同じ符号をつける.

```
def mysign(a,b)
  if b>=0 then c=(a).abs
  else c=-1*(a).abs end
  return c
end
```

tmp.rb 内の tmp(a,b)

a,b の値を入れ替える関数

```
def tmp(a,b)
  tmp=a
  a=b
  b=tmp
  return a,b
end
```

tmp.rb 内の shft(a,b,c,d)

a=b,b=c,c=d として a,b,c を返す関数

```
def shft(a,b,c,d)
  a=b
  b=c
  c=d
  return a,b,c
end
```

f1dim.rb 内の dfunc(x)

もとの関数 func(x) を微分する関数

```
def dfunc(x)
  pdx=Array.new(x)
  pdy=Array.new(x)
  df=Array.new(x)
  dd=1.0e-10
  pdx[1]+=dd
  pdy[2]+=dd
  df[1]=(func(pdx+x)-func(x))/dd
  df[2]=(func(pdy+x)-func(x))/dd
  return df
end
```

f1dim.rb 内の f1dim(x)

最小値を求める関数 f(x) を人為的に 1 次元にする関数

```
def f1dim(x)
  ncom=$ncom
  pcom=$pcom
  xicom=$xicom
  xt=Array.new(ncom+1)
  for j in 1..ncom do
    xt[j]=pcom[j]+x*(xicom[j])
  end
  #print "func(xt) = ",func(xt),"\n"
  return func(xt)
end
```

Brent.rb 内の hatena(a,b,c,d)

a,b,c,d の値を入力し，a>b ならば c を，それ以外なら d を返す関数

```
def hatena(a,b,c,d)
  if a>=b then return c
  else return d end
end
```

main.rb
CG 法を実行する．ここでは複雑な計算をせず，初期値の設定，計算時間の測定，関数 frprmn の呼び出しのみを行う．

計算時間の測定をするため現在の時間を取得するメソッド”Time.now”を用いて，最小値を求まった後の時間から求める前の時間の差を計算時間として表示する．

初期点を設定する．

```
p1=[0.0,20.0,300.0]
print "初期点","[",p1[1],",",p1[2],"]","\n"
```

#次元

```
n=2

ax=0.0
xx=1.0
cx=2.0

xfret=[0,0]

#frprmn における収束条件
$frp_ftol=1.0e-10
$frp_eps=1.0e-10

#Brent における収束条件
$Brent_tol=1.0e-5

$ncom=n
$pcom=Array.new(n+1)
$xicom=Array.new(n+1)
```

CG 法を呼び出す.

```
p frprmn(p1,n,ax,xx,cx)
```

frprmn.rb

関数 frprmn は CG 法を実行する．出発点 p1 を与えると，Fletcher-Reeves 法または Polak-Ribiere 法により関数 func を最小化する．dfunc は点 p1 での関数の勾配ベクトルを求める，関数値の収束の許容誤差を ftol におさまるまで linmin を繰り返し呼ぶ．返り値は p1 である．

初期値を設定する.

```
ftol= $frp_ftol
eps=$frp_eps
g=Array.new(n+1)
h=Array.new(n+1)
xi=Array.new(n+1)
```

```
xi=dfunc(p1)
fp=func(p1)

for j in 1..n do
  g[j]=-xi[j]
  h[j]=g[j]
  xi[j]=g[j]
end
```

linmin を呼び出し，収束判定に入るまで繰り返して最小値を求める．

```
  for its in 1..200 do
    p1,xi,xfret,ax,xx,cx=linmin(p1,xi,xfret,n,ax,xx,cx)
    if (2.0*(xfret[1]-fp).abs) <=
ftol*((xfret[1]).abs+(fp).abs+eps) then #収束判定
      return "fin."
    end
    fp=func(p1)
    xi=dfunc(p1)
    dgg=0.0;gg=0.0

    for j in 1..n do
      gg+=g[j]*g[j]
      dgg+=xi[j]*xi[j] # Fletcher-Reeves
      # dgg+=(xi[j]+g[j])*xi[j] #Polak-Ribiere
    end

    if gg==0.0 then #勾配が厳密に 0 なら完了
      print "gg-frp ",xfret,"\n"
      return xfret  #2/3
    end

    gam=dgg/gg

    for j in 1..n do
      g[j]=-xi[j]
      h[j]=g[j]+gam*h[j]
      xi[j]=g[j]+gam*h[j]
    end
```

```
  end
  return p1
```

linmin.rb

linmin は多次元の最小化と 1 次元の最小化の間に入って，最小化の補佐をする関数である．linmin は f1dim という人為的な 1 変数の関数を作り出す．この値が本来の関数 func の点 p1 を通り xi にそった直線上の値になる．その後 mnbrak と Brent を呼び出し f1dim の最小値を求める．xi は p1 が実際に動いた変位量で上書きされる．実際の最小化のルーチンは mnbrak，Brent を呼び出して行う．

初期値の設定を行う．$pcom,$xicom は func を人為的に 1 次元にする f1dim で使用する．

```
  for j in 1..n do
    $pcom[j]=p1[j]
    $xicom[j]=xi[j]
  end
  ax=0.0
  xx=1.0
```

初期値を設定した後，mnbrak，Brent を走らせて最小値を求める．p1 を xi にそって動かし p1 の値を更新する．

```
  ax,xx,cx=mnbrak(ax,xx,cx)
  xfret=Brent(ax,xx,cx,xfret)
  xmin = xfret[0]
    for j in 1..n do
    xi[j]*=xmin
    p1[j]+=xi[j]
  end
  return p1,xi,xfret,ax,xx,cx
```

mnbrak.rb

Brent 法では極小を挟むように 3 点をとらなければないないため，関数 mnbrak で初期状態の 3 点が極小を挟むようにする [2, pp.287-289].

Brent.rb

Brent 法は放物線補間と黄金分割法を用いて最小値を求める手法である．黄金分割法だけではとても効率がいいとは言えないので Brent 法は放物線補間が不適の場合に黄金分割法を使用する手法である [2, pp.289-292]3]

3.2.1 mnbrak

Ruby での mnbrak

まず，初期値を設定する．

```
gold=(1.0+sqrt(5.0))/1.5
glimit=100.0
tiny=1.0e-5
fa=f1dim(ax)
fb=f1dim(bx)
#ax,bx の役割を交換し a より b が坂の下になるようにする
if fb>fa then

  ax,bx=tmp(ax,bx)
  fa,fb=tmp(fa,fb)
end
cx=bx+gold*(bx-ax)
fc=f1dim(cx)
```

次から主ループである，囲い込みに成功するまで ($fb > fc$ の間は)while 文繰り返すようになっている．0 で割り切るのをさけるために tiny を利用する．

```
r=(bx-ax)*(fb-fc)
q=(bx-cx)*(fb-fa)
u=bx-((bx-cx)*q-(bx-ax)*r)/(2.0*mysign(fmax((q-r).abs,tiny),q-r))
ulim=(bx)+glimit*(cx-bx)
```

補外点 u が b と c の間 ($(bx-u)*(u-cx) > 0.0$) にある場合 $fu = f1dim(u)$ とし，b，c 間に極小がある ($fu < fc$) 時は $ax = bx$，$bx = u$，$fa = fb$，$fb = fu$ とし，a，u 間に極小がある ($fu > fb$) の時 $cx = u$，$fc = fu$ とし，それぞれ ax,bx，cx を返し主ループをぬける．それ以外の場合は $u = cx + gold*(cx-bx)$,$fu = f1dim$ とする．補外点 u が c と許容限界 ulim の間にある ($(cx-u)*(u-ulim) > 0.0$) 場合 $fu = f1dim(u)$ とし，さらに $fu < fc$ ならば $bx = cx$．$cx = u$，$u = cx+gold*(cx-bx)$，$fb = fc$，$fc = fu$，$fu = f1dim(u)$ とする $(u - ulim) * (ulim - cx) >= 0$ の場合補外点を最大許容限界に切り詰める．$u = ulim$，$fu = f1dim(u)$ とする．それ以外の場合 $u = cx + gold * (cx - bx)$，$fu = f1dim(u)$ とする．$ax = bx$，$bx = cx$，$cx = u$，$fa = fb$，$fb = fc$，$fc = fu$ とし以上を $fb > fc$ の間繰り返し，ax，bx，cx を返して終了する．

Maple での mnbrak

また，Ruby で作成した mnbrak を Maple でも作成し，結果を図にあらわして Ruby で作成した関数 mnbrak が正常に動いていることを確認した．

関数 fmax(a,b)

a,b を入力し，大きい方を返す関数である．

```
fmax:=proc(a,b)
  if evalf(a)>evalf(b) then
    return a;
  else return b;
  end if;
end proc:
```

関数 mysign(a,b)

a,b を入力し，b が 0 以上なら|a|を b が 0 未満なら-|a|を返す関数．a に b と同じ符号をつける．

```
mysign:=proc(a,b)
  if evalf(b)<0 then
  return -a;
    else return a;
  end if;
end proc:
```

極小を囲みたい関数 f(x) を設定し，2 点 ax,bx を適当に設定する．今回は $x^2 - 27x + 2$ の極小を挟む 3 点を求める．

```
f:=unapply(x^2-27*x+2,x);
ax:=-1;bx:=0;
```

ax,bx の役割を交換し a より b が坂の下になるようにし，他の変数についても初期値を設定する．

```
if evalf(fb)>evalf(fa) then
```

```
  tmp:=ax;ax:=bx;bx:=tmp;
  tmp:=fb;fb:=fa;fa:=tmp;
end if:

cx:=bx+gold*(bx-ax);fc:=f(cx);
```

ここで, 極小を挟むようにとりたい3点ax,bx,cxの変化をそれぞれ配列axa[],bxa[],cxa[]に保存していく.

```
axa:=Array(1..100):
bxa:=Array(1..100):
cxa:=Array(1..100):

axa[1]:=ax;bxa[1]:=bx;cxa[1]:=cx;
```

囲い込みに成功するまで ($fb > fc$ の間は)while 文繰り返す. 主ループでの細々とした計算の説明はRubyでのmnbrakで説明したので省略する. RubyでのmnbrakとMapleでのmnbrakの違いはax,bx,cxの移り変わりを知るために変化する度 (主ループを1回まわるたび) に, axa[i],bxa[i],cxa[i], にその時のax,bx,cxを格納していることである.

```
while evalf(fb)>evalf(fc) do
  r:=bx-ax*fb-fc;
  q:=bx-cx*fb-fa;
  u:=bx-((bx-cx)*q-(bx-ax)*r)/
(2.0*mysign(evalf(fmax(evalf(abs(q-r)),1.0e-20)),evalf(q-r)));
  ulim:=bx+100.0*(cx-bx);
  if evalf((bx-u)*(u-cx))>0.0 then #a
    fu:=f(u);
    if (evalf(fu)<evalf(fc)) then
      ax:=bx;
      bx:=u;
      fa:=fb;
      fb:=fu;

#ax,bx,cxの値をaxa[i],bxa[i],cxa[i]に格納
      axa[i]:=ax;bxa[i]:=bx;cxa[i]:=cx;
      break;
    elif evalf(fu)>evalf(fb) then
```

```
      cx:=u;
      fc:=fu;

#ax,bx,cx の値を axa[i],bxa[i],cxa[i] に格納
      axa[i]:=ax;bxa[i]:=bx;cxa[i]:=cx;
      break;
    end if;
    u:=cx+gold*(cx-bx);
    fu:=f(u);

#ax,bx,cx の値を axa[i],bxa[i],cxa[i] に格納
  elif evalf((cx-u)*(u-ulim)) >0.0 then    fu:=f(u);
    if evalf(fu)<evalf(fc) then
      bx:=cx;cx:=u;u:=cx+gold(cx-bx);
      fb:=fc;fc:=fu;fu:=f(u);
    end if;

#ax,bx,cx の値を axa[i],bxa[i],cxa[i] に格納
  elif evalf((u-ulim)*(ulim-cx))>0.0 then
    u:=ulim;
    fu:=f(u);

  else #d
    u:=cx+gold*(cx-bx);
    fu:=f(u);

  end if;
  ax:=bx;bx:=cx;cx:=u;
  fa:=fb;fb:=fc;fc:=fu;
  axa[i]:=ax;bxa[i]:=bx;cxa[i]:=cx;
  i:=i+1;
end do:
```

mnbrak の結果

$x^2 - 27x + 2$ での関数で最小値を挟み込んだ結果を Maple で表示する．mnbrak の主ループの繰り返し数を数えると今回の関数では 4 回繰り返して囲い込みに成功した．

初期点.

```
ppax1:=pointplot([axa[1],f(axa[1])],symbol=solidcircle,
symbolsize=20,color=red):
ppbx1:=pointplot([bxa[1],f(bxa[1])],symbol=solidcircle,
symbolsize=20,color=yellow):
ppcx1:=pointplot([cxa[1],f(cxa[1])],symbol=solidcircle,
symbolsize=20,color=green):
pf:=plot(f(x),x=-1..20):
display(ppax1,ppbx1,ppcx1,pf);
```

挟み込みが終了後の点.

```
ppax4:=pointplot([axa[4],f(axa[4])],
symbol=solidcircle,symbolsize=20,color=red):
ppbx4:=pointplot([bxa[4],f(bxa[4])],
symbol=solidcircle,symbolsize=20,color=yellow):
ppcx4:=pointplot([cxa[4],f(cxa[4])],
symbol=solidcircle,symbolsize=20,color=green):
pf:=plot(f(x),x=-1..20):
display(ppax4,ppbx4,ppcx4,pf);
```

初期点からの 3 点の変化は図 3.5 から図 3.8 の通りであり，赤色の点が ax，黄色の点が bx，緑色の点が cx である.

3.2.2 Brent

Ruby での Brent 法

初期値を以下のように設定する．ax,bx,cx は mnbrak から極小を囲むように与えられた 3 点であり，tol は精度，cgold は約 0.3819660 であり，最小値を求める上で放物線補間が不適な場合に黄金分割法を使用する.

```
  tol=$Brent_tol
  cgold=1.0-2.0/(1.0+sqrt(5.0))
  zeps=1.0e-8
  e=0.0
  d=0.0
```

#初期化

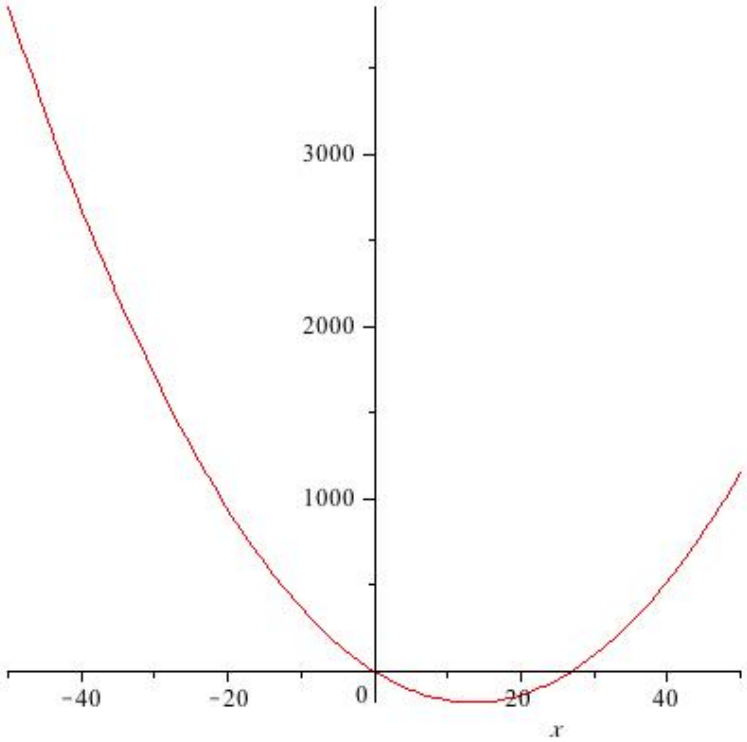

図 3.4: 極小を囲みたい関数 f(x)

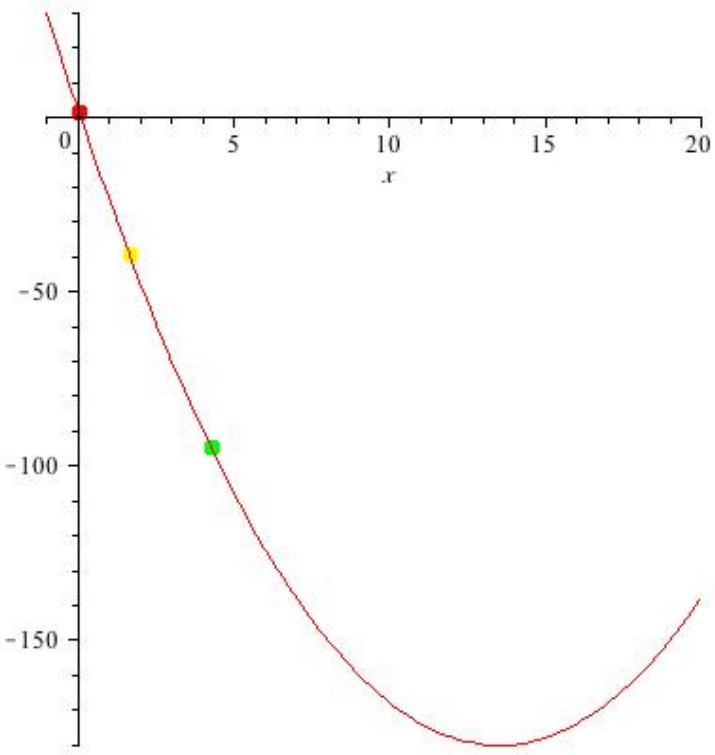

図 3.5: 囲い込みを始める前の 3 点

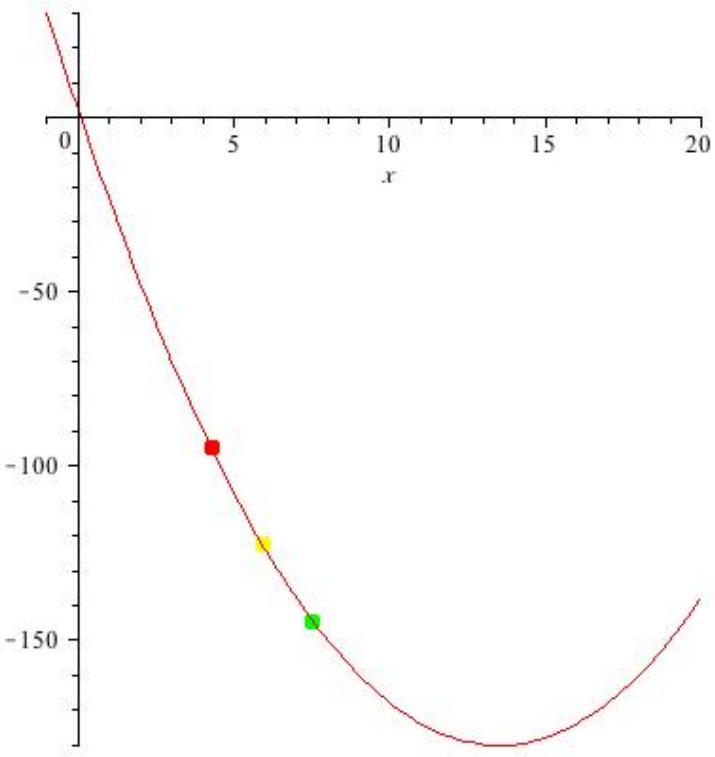

図 3.6: 主ループを 1 度まわった後の 3 点

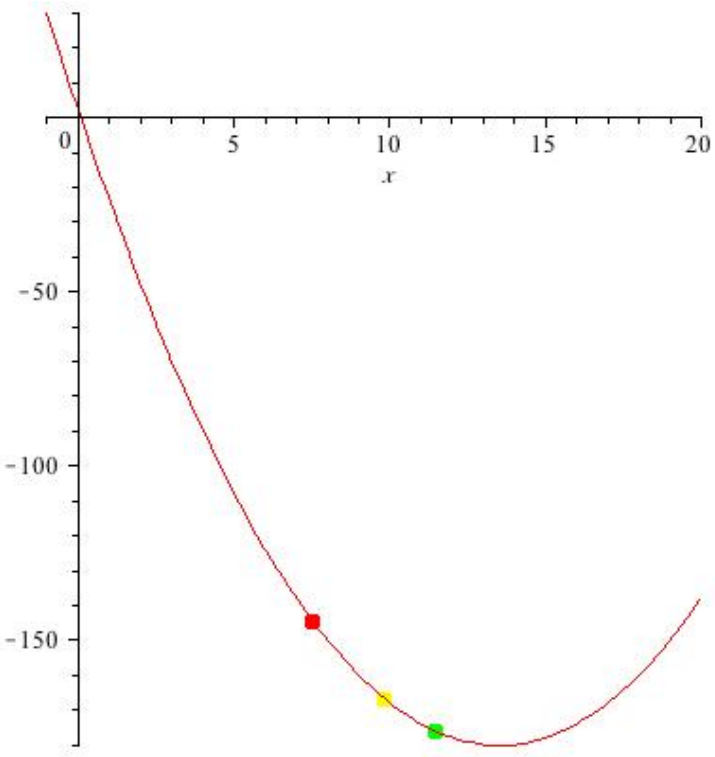

図 3.7: 主ループを 2 度まわった後の 3 点

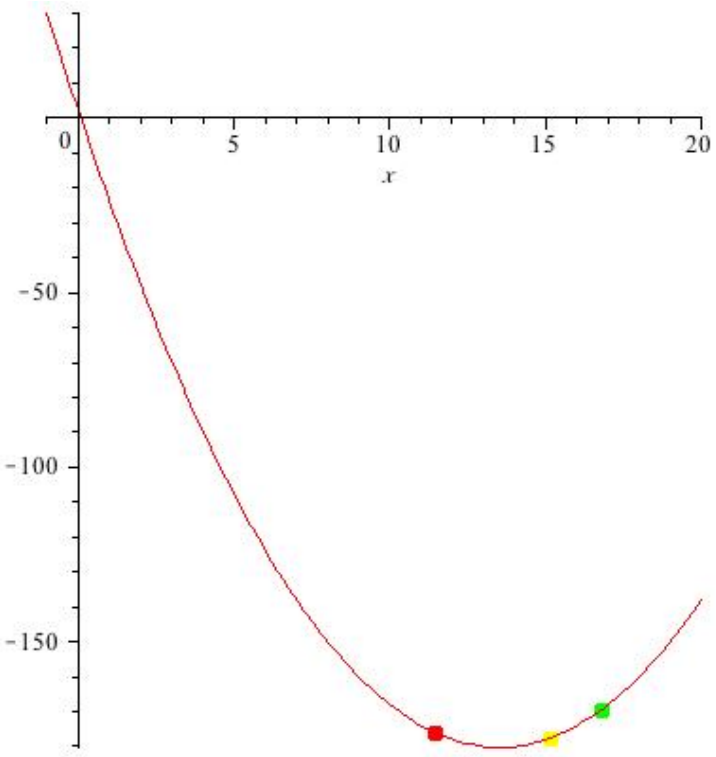

図 3.8: 囲い込み成功した後の 3 点

```
  if ax<cx then a=ax else a=cx end
  if ax>cx then b=ax else b=cx end
  x=bx;w=bx;v=bx
 # fw=fv=fx=f1dim(x)
  fw=f1dim(x)
  fv=f1dim(x)
  fx=f1dim(x)
```

まず，$|e|$ が tol1 よりも大きい場合，放物線補間をしてみる．

```
if (e).abs>tol1 then
   r=(x-w)*(fx-fv)
   q=(x-v)*(fx-fw)
   p=(x-v)*q-(x-w)*r
   q=2.0*(q-r)

   if q>0.0 then p=-p end
   q=(q).abs
   etemp=e
   e=d
end
```

その後，放物線補間が妥当かどうかの判断をする．$|p| >= |0.5 * q * etemp|$ または $p <= q * (a - x)$ または $p >= q * (b - x)$ の場合は放物線補間が不適であるので黄金分割法を使用する．

```
if (p).abs >= (0.5*q*etemp).abs || p<= q*(a-x) || p >= q*(b-x) then
  e=hatena(x,xm,a-x,b-x)
  d=cgold*e
```

その他の場合は放物線補間を採択する．

```
else
  d=p/q
  u=x+d
  if (u-a) < tol2 || (b-u)<tol2 then d=mysign(tol1,xm-x) end
end
```

以上が主ループであり，この後に続くコードは Brent で以上のことを繰り返すために値を新しくしたりするこまごまとした作業である．その作業終了後に新しくされた値で黄金分割法，放物線補間を用いて最小値探索を続ける．Brent 法は $|x - xm|$ が $tol2 - 0.5 * (b - a)$ 以下になるかまたは決められた回数主ループをまわると，最小値を求めたい関数 dunc を人為的に 1 次元にした関数 f1dim(x) の最小値とそのときの x 座標の値を返す．Brent 法はまず，極小を挟む 3 点を ax,bx,cx を入力し，$ax < cx$ なら $a = ax$，$b = cx$ それ以外の場合に $a = cx$，$b = ax$ を代入し，$x = w = v = bx$，$fw = fv = fx$ =f1dim(x) と初期値を設定する．Brent 法は 1 次元の最小化を求める手法なので与えられた最小値を求めたい関数 func(x) を 1 次元化したものが f1dim(x) である

Maple での Brent 法

3.2.3 Brent の結果

Ruby で $f(x) = x^2 - 27x + 2$ の最小値を求めた．この関数の最小値は (x,f(x))=(13.50, -180.25) である．結果は fret=[13.49999999, -180.2500000] でありほぼ正確に作動していることがわかる．mnbrak 同様に Brent 法で求めた最小値の妥当性を検討するために Maple でも Brent 法を作成し Ruby で最小値を求めた関数と同じ関数 $f(x) = x^2 - 27x + 2$ 最小値を求め，出てきた答えを図 3.9 に示した．赤い点は求められた最小値である．

```
f:=unapply(x^2-27*x+2,x);
```

この関数の最小値は (x,f(x))=(13.50, -180.25) である．図 3.9 は Maple での結果を図に表した．

関数 hatena(a,b,c,d)

a,b を入力し，a>b なら c を，それ以外の場合には d を返す関数.

```
hatena:=proc(a,b,c,d)
  if evalf(a)>=evalf(b) then return c;
  else return d;
  end if;
end proc:
```

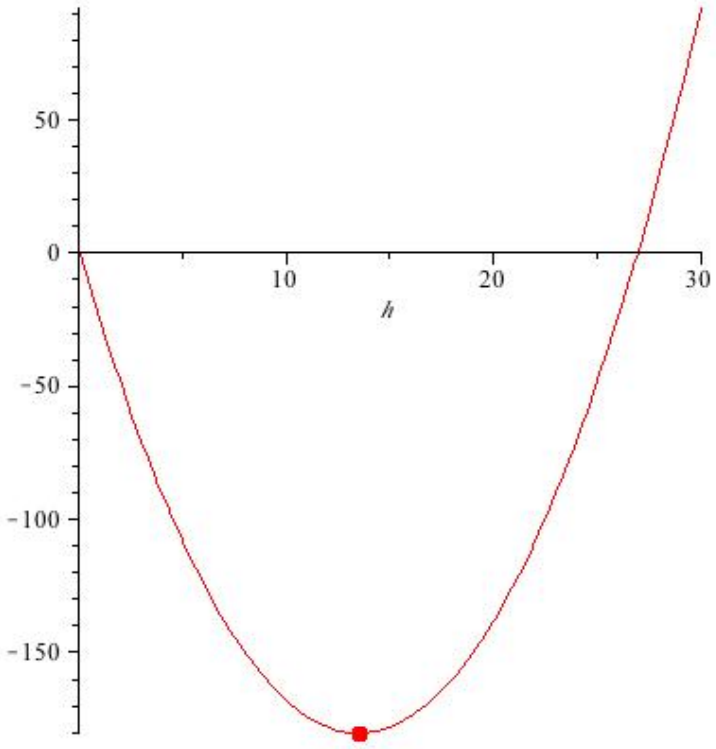

図 3.9: Maple での Brent 法の結果

関数 mysign(a,b)

a,b を入力し，b が 0 以上なら|a|を b が 0 未満なら-|a|を返す関数．a に b と同じ符号をつける．

```
mysign:=proc(a,b)
  if evalf(b)<0 then
  return -a;
    else return a;
  end if;
end proc:
```

初期値の設定をする．

```
tol:=1.0e-10;
cgold:=1.0-(sqrt(5.0)-1)/2;
zeps:=1.0e-10;
e:=0.0;d:=0.0;
ax:=-20.0;bx:=0.0;cx:=20.0;
if evalf(ax) < evalf(cx) then
  a:=ax;
else a:=cx;
end if;
if evalf(ax)>evalf(cx) then
  b:=ax;
else
  b:=cx;
end if;

x:=bx; w:=bx; v:=bx;
fw:=f(x);fv:=f(x);fx:=f(x);
```

以下から主ループである．

```
   xm:=0.5*(a+b);
   tol1:=tol*abs(x)+zeps;
   tol2:=2.0*tol1;
```

```
#収束判定
if evalf(abs(x-xm)) <= evalf(tol2-0.5*(b-a)) then
  xmin:=x;
  fret[1]:=xmin;
  fret[2]:=fx;
  printf("fin");
  break;
end if;
```

収束判定で一定の値よりも誤差が小さくなるとループを抜け，終了する．その際，最小値 fx を fret[2] に，その時の x 座標 xmin を fret[1] に格納する．
主ループでの複雑な計算の説明は先に述べたので省略する．主ループを適当な回数 (例えば 100 回) 繰り返して，その間にに収束判定に入らなければ，ループを脱出し，その際の最小値 fx を fret[2] に，その時の x 座標 xmin を fret[1] に格納する．

```
if evalf(abs(e))>evalf(tol1) then
  r:=(x-w)*(fx-fv);
  q:=(x-v)*(fx-fw);
  p:=(x-v)*q-(x-w)*r;
  q:=2.0*(q-r);

  if evalf(q)>0.0 then
    p:=-p;
  end if;

  q:=abs(q);
  etemp:=e;
  e:=d;
  if evalf(abs(p)) >= evalf(abs(0.5*q*etemp)) or
 evalf(p)<= evalf(q*(a-x)) or evalf(p) >= evalf(q*(b-x)) then
    e:=hatena(evalf(x),evalf(xm),a-x,b-x);
    d:=cgold*e;
  else
    d:=p/q;
    u:=x+d;
    if evalf((u-a)) < evalf(tol2) or evalf((b-u))<evalf(tol2) then
      d:=mysign(evalf(tol1),evalf(xm-x));
    end if;
  end if;
```

```
else
  e:=hatena(evalf(x),evalf(xm),a-x,b-x);
  d:=cgold*e;
end if;

u:=hatena(evalf(abs(d)),evalf(tol1),x+d,x+mysign(evalf(tol1),evalf(d)));
fu:=f(u);
if evalf(fu)<=evalf(fx) then
  if evalf(u)>=evalf(x) then
    a:=x;
  else
    b:=x;
  end if;
  v:=w; w:=x; x:=u;
  fv:=fw; fw:=fx; fx:=fu;
else
  if evalf(u)<evalf(x) then
    a:=u;
  else
    b:=u;
 end if;
  if evalf(fu)<=evalf(fw) or w=x then
    v:=w; w:=u; fv:=fw; fw:=fu;
  elif evalf(fu) <= evalf(fv) or v=x or v=w then
    v:=u; fv:=fu;
  end if;
end if;
```

3.2.4 CG 法で求めた最小値

Ruby で作成した CG 法で linmin が 1 回繰り返されるたびに，つまり人為的に 1 次元にした関数の最小値が求まり，次の点へ移動するたびに配列 p1 を出力し p1 の変化を Maple で図にした．p1 は要素数 3 の配列であり，p1[0] はダミー，p1[1] は x 座標，p1[2] は y 座標である．

図 3.10 は関数 $f(x,y) = (x-1.0)^2 + (y-2.0)^2 + \frac{xy(x^2-y^2)}{x^2+y^2}$ と初期点，$p1 = [0.0, 20.0, 300.0]$ の変化，CG 法で求めた最小値を Maple で出力した図である．初期点を茶色い四角で，途中の p1 を黄色い丸で，最小値を赤い丸で示した．図 3.11

は，最小値を求めたい関数 f(x,y) を等高線であらわし，そこへ p1 の変化を線で結んだものである．直線の角が p1 の値であり，Brent 法で人為的に変換した 1 次の関数の最小値が求まると次の方向へほぼ直角に方向を変え，きれいに最小値に収束していく様子がわかる．

```
dr:=300  #プロットする範囲
pf:=plot3d(f(x,y),x=-dr..dr,y=-dr..dr,axes=box): #\ref{cg3d}作成
p1:=[20.0,300.0]:  #初期点
ki1:=[98.374997014778, 122.42182806961]:  #以下，linmin が終了し新しい
p1
ki2:=[-51.3577433177724, 38.1049068333855]:
.
.
.
ki23:=[1.19135715253226, 2.72547932363407]:
ki24:=[1.19135318379068, 2.72547649814474]:  #最小値
```

$f(x,y)$ を 3 次元で表した図と同じ図に p1,ki1,...,ki24 を表示するために以下のことをする．
p1 の pointplot．点のサイズは 20，色はブラウンである．op は配列の要素を取り出す．

```
pp1:=pointplot3d([op(p1),f(o(p1))],
symbol=solidbox,symbolsize=20,color=brown):
```

ki1,ki2 の pointplot．ki23 まで同様にする．点のサイズは 15，色はイエローである．

```
pki1:=pointplot3d([op(ki1),f(op(ki1))],
symbol=solidcircle,symbolsize=15,color=yellow):
pki2:=pointplot3d([op(ki2),f(op(ki2))],symbol=solidcircle,
symbolsize=15,color=yellow):
```

最小値 ki24 での pointplot．わかりやすい様に点のサイズを 25，色をレッドにした．

```
pki24:=pointplot3d([op(ki24),f(op(ki24))],
symbol=solidcircle,symbolsize=25,
color=red):
```

以上を display を使い，f(x,y)，p1 を同じ図に表示する
display(pf,pp1,pki1,pki2,pki3,pki4,pki5,pki6,pki7,pki8,pki9,pki10,pki11,pki12,
pki13,pki14,pki15,pki16,pki17,pki18,pki19,pki20,pki21,pki22,pki23,pki24);

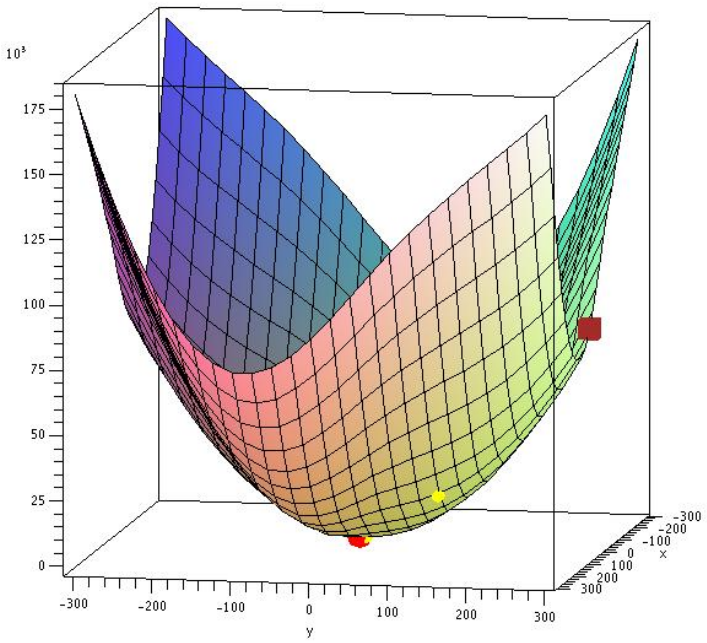

図 3.10: f(x,y) と p1 の変化

計算結果

$f(x,y) = (x-1.0)^2+(y-2.0)^2+\frac{xy(x^2-y^2)}{x^2+y^2}$，を初期点 $p1 = [0.0, 20.0, 300.0]$，の最小値を CG 法で求めた場合場合，最小値は (x,y)=(1.19135318379068, 2.72547649814474) であり．計算時間は 0.014771 秒であった．

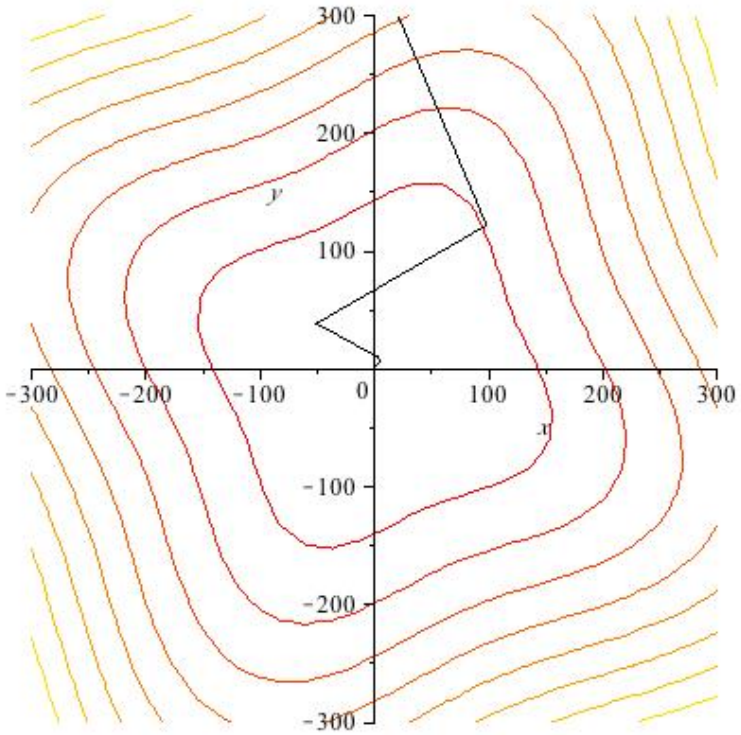

図 3.11: f(x,y) の等高線図と p1 の変化

3.3 Powell法

Powell 法で使用するプログラムはすでに作成された CG 法とほぼかわらない．mnbrak や Brent の中身は 3.2 で述べたので省略する．Powell 法でも linmin を使用するが，Powell 法では関数 linmin の引数に xfret を使用せず，返り値に xfret ではなく xfret[1] つまり，Brent で求められた人為的な 1 次元の関数の最小値のみを返り値とする．
Powell で使用する関数を sqr を説明する

Powell.rb 内の sqr(a)

a を入力し，a=0.0 なら 0.0 を，それ以外なら a^2 を返す関数．これは，ルートの中が負の値になった場合の対策である．

```
def sqr(a)
  if a==0.0 then return 0.0
  else return a*a
  end
end
```

3.3.1 Powell法で求めた最小値

CG 法同様 Ruby で作成した linmin が 1 回繰り返されるたびに，つまり人為的に 1 次元にした関数の最小値が求まり，次の点へ移動するたびに配列 p1 を出力し p1 の変化を Maple で図にした．p1 は要素数 3 の配列であり，p1[0] はダミー，p1[1] は x 座標，p1[2] は y 座標である．初期点 p1，最小値を求める関数．計算精度は CG 法と同じである．
図 3.12 は関数 $f(x,y)=(x-1.0)^2+(y-2.0)^2+\frac{xy(x^2-y^2)}{x^2+y^2}$ と初期点，$p1=[0.0,20.0,300.0]$(p1[1] はダミー) の変化，CG 法で求めた最小値を Maple で出力した図である．初期点を茶色い四角で，途中の p1 を黄色い丸で，最小値を赤い丸で示した．
図 3.13 は，最小値を求めたい関数 $f(x,y)$ を等高線であらわし，そこへ p1 の変化を線で結んだものである．直線の角が p1 の値であり，Brent 法で人為的に変換した 1 次の関数の最小値が求まると次の方向へほぼ直角に方向を変え，きれいに最小値に収束していく様子がわかる．

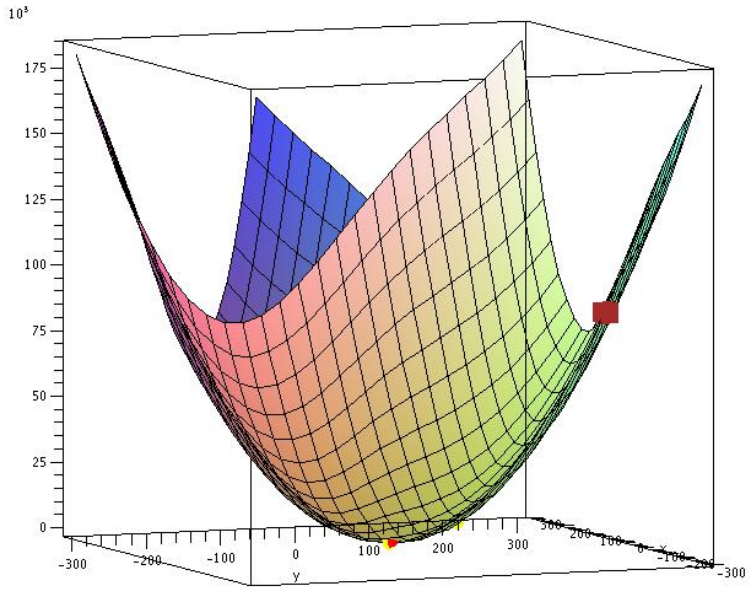

図 3.12: f(x,y) と p1 の変化

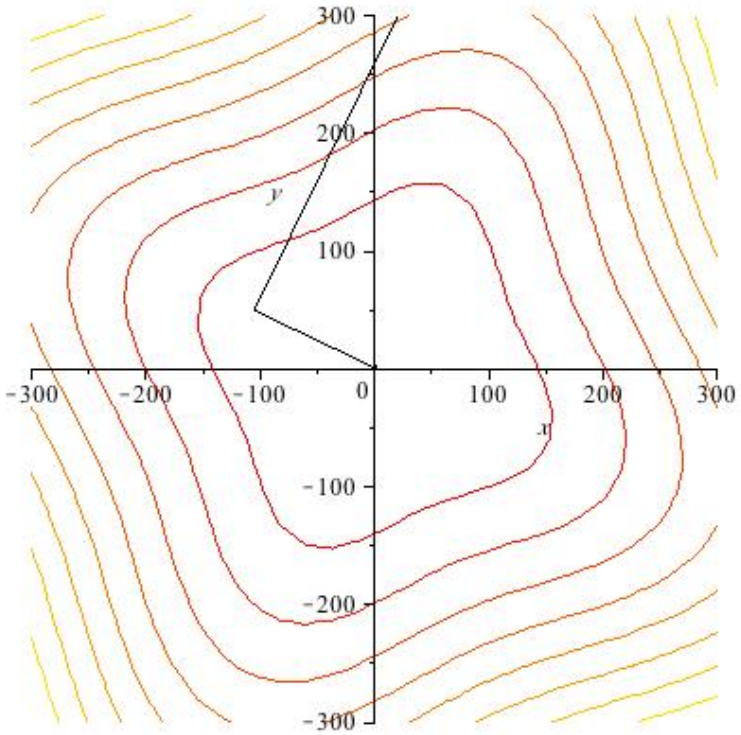

図 3.13: f(x,y) の等高線図と p1 の変化

計算結果

$f(x,y)=(x-1.0)^2+(y-2.0)^2+\frac{xy(x^2-y^2)}{x^2+y^2}$，を初期点 $p1=[0.0,20.0,300.0]$，からスタートした場合，最小値は (x,y)=(1.19135311867942, 2.72547558803349) であり．0.050876 秒で求められた．まず，初期値を設定する．

```
itmax=200
pt=Array.new(n+1)
ptt=Array.new(n+1)
xi_a=[]
xi_b=[]
xi_a[0]=0.0
xi_b[0]=0.0
xit=[0.0,0.0,0.0]
xi=[[0,0,0],[0,10,15],[0,20,25]]
#ここで xi 作成終了
xfret=func(p1)
ftol= $frp_ftol
eps=$frp_eps
```

次に初期点を保存する．

```
for j in 1..n do
  pt[j]=p1[j]
end
```

以下より主ループである．

```
fp=xfret
ibig=0
del=10e-10  #関数値の最大限少量を求める変数 del
for i in 1..n do  #各反復で方向集合の全要素についてループする
   for j in 1..n do
    xit[j]=xi[j][i]  #方向をコピー
  end
  fptt=xfret
  p1,xit,xfret,ax,xx,cx=linmin(p1,xit,n,ax,xx,cx)
  if (fptt-xfret).abs>del then
    del=(fptt-xfret).abs  #最大の減少であれば記録する
    ibig=i
```

```
  end
end
```

Powell 法の収束判定

```
if 2.0*(fp-xfret).abs <= ftol*((fp).abs+(xfret).abs) then
  #収束判定
  p "fin"
  return p1
end
```

あらかじめ決められた回数以上繰り返しても最小値が求まらなかったらエラーと表示

```
if (iter==itmax) then
  print "error"
end
```

補外点と平均移動方向を求め，古い出発点を保存する．

```
for m in 1..n do
  ptt[m]=2.0*p1[m]-pt[m]
  xit[m]=p1[m]-pt[m]
  pt[m]=p1[m]
end


fptt=func(ptt)
```

新しい方向の最小まで移動し，新しい方向を保存する．

```
if fptt < fp then
  t=2.0*(fp-2.0*xfret+fptt)*sqrt(fp-xfret-del)-del*sqrt(fp-fptt)
  if t<0.0 then
    p1,xit,xfret,ax,xx,cx=linmin(p1,xit,n,ax,xx,cx)
    for l in 1..n do
      xi[l][ibig]=xi[l][n]
      xi[l][n]=xit[l]
    end
```

```
  end
end
```

第4章　総括

MC 法は一般的にも計算時間も精度もあまりよくないと言われていが複雑ではない 2 次関数ではほぼ正確に最小値を導きだした．本研究では複雑な 2 次関数に関しては，MC 法で最小値を求められるかどうかの検証を行っていないため検証の余地がある．Powell 法も CG 法も複雑な関数においてほぼ正確に最小値が求めることができ，精度には大きな差はなかった，また，Powell 法は CG 法に比べて主ループを繰り返す回数が少ないので，勾配ベクトルを使わない分無駄の多い手法の割には収束が早かったが，計算時間は CG 法の約 3.4 倍であったことから，勾配ベクトルを使用した場合と使用しない場合では計算時間に大きく差が出ることがわかった．

引用文献

[1] 西谷滋人,『個体物理の基礎』, (森北出版株式会社 2006)　pp.109-115

[2] William H. Press 他,『ニューメリカルレシピ・イン・シー』, (技術評論社 1993)
pp.283-338

[3] 菅沼義昇, システムエンジニアの基礎知識,
http://www.sist.ac.jp/~suganuma/kougi/other_lecture/SE/opt/nonlinear/nonlinear.h

謝辞

本研究を遂行するにあたり，終始多大なるご指導，及び丁寧な助言を頂いた西谷滋人教授に深く感謝の意を表します.

また，日常の議論を通じて多くの知識や示唆を頂いた西谷研究室の皆様に感謝します.

付 録A　MC法

A.1　TSP

A.1.1　MapleでのTSP

```
restart;

with(plots):
with(plottools):
with(stats):
with(LinearAlgebra):
with(linalg):
with(ListTools):
with(combinat,permute):
with(Statistics):

Distance:=proc(a,b)
return sqrt((Position[a][1]-Position[b][1])^2+
(Position[a][2]-Position[b][2])^2);
end proc:

E:=proc(p)
local S:=0,i;
 for i from 1 to N_city do
  S:=S+dis[p[i],p[i+1]];
 end do;
 return S;
end proc:

N_city:=10;
Position:=seq([evalf(rand()/10^12),evalf(rand()/10^12)],i=1..N_city);
Path:=[seq(i,i=1..N_city),1];
Real_Path:=[seq(Position[i],i=1..N_city),Position[1]];
```

```
PLOT(CURVES(Real_Path));

for i from 1 to N_city-1 do
Distance(i,i+1);
end do;
Distance(N_city,1):

dis:=array(1..N_city,1..N_city);
for i from 1 to N_city do
 for s from 1 to N_city do
  dis[i,s]:=Distance(i,s);
 end do;
end do;
print(dis);

E(Path);

da:=proc(a,b)
  local t;
  global Path;
  t:=Path[a];
  Path[a]:=Path[b];
  Path[b]:=t;
  return Path;
end proc;
proc(a, b)  ...  end;

a:=Path;

E_trace:=[E(Path)];

T:=0.02;
pb_rd:=rand(0..100000):
sel_city:=rand(2..N_city):

for i from 1 to 10000 do
  s1:=sel_city();
  s2:=sel_city():
  dela:=da(s1,s2);
```

```
  dE:=E(dela)-E(a);

  if (dE<=0) then
    a:=dela;
    E_trace:=[op(E_trace),E(a)];
  elif (dE>0) then
    pb:=exp(-dE/T);
    rd:=pb_rd()*10^(-5);
    if (rd<pb) then
      a:=dela;
      E_trace:=[op(E_trace),E(a)];
    end if;
  end if;
end do:
E(a);
a;
E_trace;

RP:=[seq(Position[a[i]],i=1..N_city),Position[1]]:
PLOT(CURVES(RP));

listplot(E_trace);

for i from 1 to 10 do
rd:=evalf(pb_rd()*10^(-5));
end do;
pb:=exp(-dE/0.5);
pb*10000;
```

A.1.2 Ruby での TSP

```
require 'pp'
include Math
require "Matrix"

N_city=8
Position_x=[]
Position_y=[]
```

```
Path=[]
Position = []
$dis={ }
k=0
f=0
T=0.02

#a,b 間の距離を計算する関数
def Distance(a,b)
  return sqrt( ((Position[a][0]-Position[b][0])**2)+
((Position[a][1]-Position[b][1])**2))
end

#総距離を計算
def E(a)
  s = 0
  for i in 0..N_city-1
      s = s + $dis[[a[i],a[i+1]]]
  end
  return s
end

#巡回する都心の座標をランダムに作成
for i in 0..N_city-1
  Position_x[i]=rand()
  Position_y[i]=rand()
  Position << [Position_x[i],Position_y[i]]
end

#まわる都市の順番を Path に入れる (最初は 0,1,..,N_city-1,0)
for i in 0..N_city-1
  Path[i]=i
end
Path[N_city]=0

p Path

#それぞれの都市間の距離を計算し$dis[] に格納
for i in 0..N_city-1
```

```
  for j in 0..N_city-1
  $dis[[i,j]] = Distance(i,j)
#  print(i,",",j,"=",$dis[[i,j]],"\n")   それぞれの都市間の距離を表示
  end
end

#最適化前のトータルコスト
p E(Path)

#Path の複製 dela[] をつくる
a = Path
dela = []
a.each do |ai|
  dela << ai
end

#コストを計算し dela を変更していく
e_trace=[]
300.times do
  s1=rand(N_city-1)+1
  s2=rand(N_city-1)+1

  dela = []
  a.each do |ai| dela << ai end

  f=dela[s1]
  dela[s1]=dela[s2]
  dela[s2]=f

  dE=E(dela)-E(a)

  if dE<0 then
    a=dela
    e_trace << E(a)
  elsif dE>0 then
    pb=exp(-dE/T)
    rd=rand(10000000)*(10**-7)
    if rd<pb then
      a=dela
```

```
      e_trace << E(a)
    end
  end
end

#最適化後のトータルコスト
p E(a)

#最適化後のルート
p a
```

付 録B　CG法

B.1　RubyでのCG法

B.1.1　main.rb

```
require "./frprmn.rb"
require 'pp'

t0 = Time.now
#初期点p1[dummy,x,y]
p1=[0.0,20.0,300.0]
print "初期点","[",p1[1],",",p1[2],"]","\n"

#次元
n=2

ax=0.0
xx=1.0
cx=2.0

xfret=[0,0]

#frprmnにおける収束条件
$frp_ftol=1.0e-10
$frp_eps=1.0e-10

#Brentにおける収束条件
$Brent_tol=1.0e-5

$ncom=n
$pcom=Array.new(n+1)
$xicom=Array.new(n+1)
```

```
p frprmn(p1,n,ax,xx,cx)
t1 = Time.now
print (t1-t0,"s\n")
```

B.1.2 frprmn.rb

```
require 'pp'
include Math
require "linmin.rb"
require "etc.rb"
require "f1dim.rb"

def frprmn(p1,n,ax,xx,cx)
  ftol= $frp_ftol
  eps=$frp_eps
  print "ax,xx,cx=",ax,"\t",xx,"\t",cx,"\n"
  g=Array.new(n+1)
  h=Array.new(n+1)
  xi=Array.new(n+1)
  xi=dfunc(p1)
  fp=func(p1)
  #fp=f1dim(p1) #####

  for j in 1..n do
    g[j]=-xi[j]
    h[j]=g[j]
    xi[j]=g[j]
  end

  for its in 1..200 do
    p1,xi,xfret,ax,xx,cx=linmin(p1,xi,xfret,n,ax,xx,cx) #2/3
    print "[",p1[1],", ",p1[2],"]","\n"
    if (2.0*(xfret[1]-fp).abs) <= ftol*((xfret[1]).abs+(fp).abs+eps) then
      return "fin."
    end
    fp=func(p1)
    xi=dfunc(p1)
    dgg=0.0;gg=0.0
```

```
    for j in 1..n do
      gg+=g[j]*g[j]
      dgg+=xi[j]*xi[j] # Fletcher-Reeves
      # dgg+=(xi[j]+g[j])*xi[j] #Polak-Ribiere
    end

    if gg==0.0 then
      print "gg-frp ",xfret,"\n"
      return xfret  #2/3
    end
    gam=dgg/gg
    for j in 1..n do
      g[j]=-xi[j]
      h[j]=g[j]+gam*h[j]
      xi[j]=g[j]+gam*h[j]
    end
  end
  return p1
end
```

B.1.3 linmin.rb

```
require 'pp'
include Math
require "mnbrak.rb"
require "Brent.rb"

def linmin(p1,xi,xfret,n,ax,xx,cx)
 # nrfunc=func
  for j in 1..n do
    $pcom[j]=p1[j]
    $xicom[j]=xi[j]
  end
  ax=0.0
  xx=1.0
  ax,xx,cx=mnbrak(ax,xx,cx)
  xfret=Brent(ax,xx,cx,xfret)
  xmin = xfret[0]
    for j in 1..n do
```

```
    xi[j]*=xmin
    p1[j]+=xi[j]
  end
  return p1,xi,xfret,ax,xx,cx  #2/3
end
```

B.1.4 mnbrak.rb

```
include Math
require 'pp'
require "f1dim.rb"
require "etc.rb"
require "tmp.rb"

def mnbrak(ax,bx,cx)
#  print "start:ax,bx,xx=",ax,"\t",bx,"\t",cx,"\n"
  gold=(1.0+sqrt(5.0))/1.5
  glimit=100.0
  tiny=1.0e-5

  fa=f1dim(ax)
  fb=f1dim(bx)
  #ax,bx の役割を交換し a より b が坂の下になるようにする
#  print "fb-fa = ",fb-fa,"\n"
  if fb>fa then
    ax,bx=tmp(ax,bx)
    fa,fb=tmp(fa,fb)
  end
  #c の初期設定
  cx=bx+gold*(bx-ax)
  fc=f1dim(cx)
  while fb>fc do
    r=(bx-ax)*(fb-fc)
    q=(bx-cx)*(fb-fa)
    u=bx-((bx-cx)*q-(bx-ax)*r)/(2.0*mysign(fmax((q-r).abs,tiny),q-r))
    ulim=(bx)+glimit*(cx-bx)
    if  (bx-u)*(u-cx) > 0.0 then
      fu=f1dim(u)
```

```
      if fu<fc then #b,c 間に極小あり
        ax=bx
        bx=u
        fa=fb
        fb=fu
        return ax, bx, cx # return 2011/01/31 に変更
      elsif fu>fb then #a,u 間に極小あり
        cx=u
        fc=fu
        return ax, bx, cx
      end
      u=cx+gold*(cx-bx)
      fu=f1dim(u)
    elsif (cx-u)*(u-ulim)>0.0 then
#print "ax,bx,cx= ",ax,"\t",bx,"\t",cx,"\n"
      fu=f1dim(u)
      if fu<fc then
        bx,cx,u=shft(bx,cx,u,cx+gold*(cx-bx))
        fb,fc,fu=shft(fb,fc,fu,f1dim(u))
      end
#print "ax,bx,cx= ",ax,"\t",bx,"\t",cx,"\n"
    elsif (u-ulim)*(ulim-cx)>=0.0 then
      u=ulim
      fu=f1dim(u)
    else
      u=cx+gold*(cx-bx)
      fu=f1dim(u)
    end
    ax,bx,cx=shft(ax,bx,cx,u)
    fa,fb,fc=shft(fa,fb,fc,fu)
    end
  return ax,bx,cx
end
```

B.1.5 Brent.rb

```
include Math
require "tmp.rb"
```

```
require "f1dim.rb"
require "etc.rb"

def hatena(a,b,c,d)
  if a>=b then return c
  else return d end
end

def Brent(ax,bx,cx,xfret)
  tol=$Brent_tol
  cgold=1.0-(sqrt(5.0)-1.0)/2.0
  zeps=1.0e-8
  e=0.0
  d=0.0

#初期化
  if ax<cx then a=ax else a=cx end
  if ax>cx then b=ax else b=cx end
  x=bx;w=bx;v=bx
 # fw=fv=fx=f1dim(x)
  fw=f1dim(x)
  fv=f1dim(x)
  fx=f1dim(x)

  for j in 1..100 do
    xm=0.5*(a+b)
    tol1=tol*(x).abs+zeps
    tol2=2.0*tol1

    #収束判定
    if (x-xm).abs<=tol2-0.5*(b-a) then
      xmin=x
       return [xmin,fx]
    end

    if (e).abs>tol1 then
      r=(x-w)*(fx-fv)
      q=(x-v)*(fx-fw)
      p=(x-v)*q-(x-w)*r
```

```
      q=2.0*(q-r)

      if q>0.0 then p=-p end
      q=(q).abs
      etemp=e
      e=d
      if (p).abs >= (0.5*q*etemp).abs || p<= q*(a-x) || p >= q*(b-x) then
        e=hatena(x,xm,a-x,b-x) #x>=xm:e=a-x, x<xm:e=b-x
        d=cgold*e
      else
        d=p/q
        u=x+d
        if (u-a) < tol2 || (b-u)<tol2 then d=mysign(tol1,xm-x) end
      end
    else
      e=hatena(x,xm,a-x,b-x) #x>=xm:e=a-x, x<xm:e=b-x
      d=cgold*e
    end
    u=hatena((d).abs,tol1,x+d,x+mysign(tol1,d))
    fu=f1dim(u)
    if fu<=fx then
      if u>=x then a=x
      else b=x end
      v,w,x=shft(v,w,x,u)
      fv,fw,fx=shft(fv,fw,fx,fu)
    else
      if u<x then a=u else b=u end
      if fu<=fw || w==x then v=w;w=u;fv=fw;fw=fu
      elsif fu <= fv || v==x || v==w then v=u;fv=fu end
    end
  end
  xmin=x
  print("bret_ERROR ",xmin,"\t",fx,"\n");
  return [xmin,fx] #[極小値, f(極小値)]
end
```

B.1.6 f1dim.rb

```
include Math
require 'pp'

#関数 func
def func(a)
  x=a[1]
  y=a[2]
  f1=(x-1.0)**2+(y-2.0)**2
  f2=x*y*(x**2-y**2)/(x**2+y**2)
  return f1+f2
end

#関数 func を微分
def dfunc(x)
  pdx=Array.new(x)
  pdy=Array.new(x)
  df=[0.0,0.0,0.0]
  dd=1.0e-10
  pdx[1]+=dd
  pdy[2]+=dd
  df[1]=-((func(pdx+x)-func(x))/dd)
  df[2]=-((func(pdy+x)-func(x))/dd)
  return df
end

#関数 func を 1 次元に変換
def f1dim(x)
  ncom=$ncom
  pcom=$pcom
  xicom=$xicom
  xt=Array.new(ncom+1)
  j=1
  for j in 1..ncom do
    xt[j]=pcom[j]+x*(xicom[j])
  end
  return func(xt)
end
```

B.1.7 etc.rb

```
def fmax(a,b)
  if a>=b then return a
  else return b end
end

def mysign(a,b)
  if b>=0 then c=(a).abs
  else c=-1*(a).abs end
  return c
end
```

B.1.8 tmp.rb

```
def tmp(a,b)
  tmp=a
  a=b
  b=tmp
  return a,b
end

def shft(a,b,c,d)
  a=b
  b=c
  c=d
  return a,b,c
end
```

B.1.9 Mapleでのmnbrak，Brent

mnbrak.mw

```
restart;
with(combinat,permute):
```

```
with(plots):
with(stats):
with(LinearAlgebra):
with(linalg):

mysign:=proc(a,b)
  if evalf(b)<0 then
  return -a;
    else return a;
  end if;
end proc:

fmax:=proc(a,b)
  if evalf(a)>evalf(b) then
    return a;
  else return b;
  end if;
end proc:
gold:=(1+sqrt(5))/2;

axa:=Array(1..100):
bxa:=Array(1..100):
cxa:=Array(1..100):

f:=unapply(x^2-27*x+2,x);

plot(f(x),x=-50..50);

ax:=-1;bx:=0;

fa:=f(ax);fb:=f(bx);

if evalf(fb)>evalf(fa) then
  tmp:=ax;ax:=bx;bx:=tmp;
  tmp:=fb;fb:=fa;fa:=tmp;
end if:
cx:=bx+gold*(bx-ax);fc:=f(cx);

evalf(fc);
```

```
axa[1]:=ax;bxa[1]:=bx;cxa[1]:=cx;

i:=1;

evalf(fb-fc);
while evalf(fb)>evalf(fc) do
  r:=bx-ax*fb-fc;
  q:=bx-cx*fb-fa;
  u:=bx-((bx-cx)*q-(bx-ax)*r)/
(2.0*mysign(evalf(fmax(evalf(abs(q-r)),1.0e-20)),evalf(q-r)));
  ulim:=bx+100.0*(cx-bx);
  if evalf((bx-u)*(u-cx))>0.0 then #a
    fu:=f(u);
    if (evalf(fu)<evalf(fc)) then
      ax:=bx;
      bx:=u;
      fa:=fb;
      fb:=fu;
      axa[i]:=ax;bxa[i]:=bx;cxa[i]:=cx;
      break;
    elif evalf(fu)>evalf(fb) then
      cx:=u;
      fc:=fu;
      axa[i]:=ax;bxa[i]:=bx;cxa[i]:=cx;
      break;
    end if;
    u:=cx+gold*(cx-bx);
    fu:=f(u);

  elif evalf((cx-u)*(u-ulim)) >0.0 then #b
    fu:=f(u);
    if evalf(fu)<evalf(fc) then
      bx:=cx;cx:=u;u:=cx+gold(cx-bx);
      fb:=fc;fc:=fu;fu:=f(u);
    end if;

  elif evalf((u-ulim)*(ulim-cx))>0.0 then #c
    u:=ulim;
```

```
    fu:=f(u);

  else #d
    u:=cx+gold*(cx-bx);
    fu:=f(u);

  end if;
  ax:=bx;bx:=cx;cx:=u;
  fa:=fb;fb:=fc;fc:=fu;
  axa[i]:=ax;bxa[i]:=bx;cxa[i]:=cx;
  i:=i+1;
end do:
i;

ppax1:=pointplot([axa[1],f(axa[1])],
symbol=solidcircle,symbolsize=20,color=red):
ppbx1:=pointplot([bxa[1],f(bxa[1])],
symbol=solidcircle,symbolsize=20,color=yellow):
ppcx1:=pointplot([cxa[1],f(cxa[1])],
symbol=solidcircle,symbolsize=20,color=green):
pf:=plot(f(x),x=-1..20):
display(ppax1,ppbx1,ppcx1,pf);


ppax2:=pointplot([axa[2],f(axa[2])],
symbol=solidcircle,symbolsize=20,color=red):
ppbx2:=pointplot([bxa[2],f(bxa[2])],
symbol=solidcircle,symbolsize=20,color=yellow):
ppcx2:=pointplot([cxa[2],f(cxa[2])],
symbol=solidcircle,symbolsize=20,color=green):
pf:=plot(f(x),x=-1..20):
display(ppax2,ppbx2,ppcx2,pf);


ppax3:=pointplot([axa[3],f(axa[3])],
symbol=solidcircle,symbolsize=20,color=red):
ppbx3:=pointplot([bxa[3],f(bxa[3])],
symbol=solidcircle,symbolsize=20,color=yellow):
ppcx3:=pointplot([cxa[3],f(cxa[3])],
```

```
symbol=solidcircle,symbolsize=20,color=green):
pf:=plot(f(x),x=-1..20):
display(ppax3,ppbx3,ppcx3,pf);


ppax4:=pointplot([axa[4],f(axa[4])],
symbol=solidcircle,symbolsize=20,color=red):
ppbx4:=pointplot([bxa[4],f(bxa[4])],
symbol=solidcircle,symbolsize=20,color=yellow):
ppcx4:=pointplot([cxa[4],f(cxa[4])],
symbol=solidcircle,symbolsize=20,color=green):
pf:=plot(f(x),x=-1..20):
display(ppax4,ppbx4,ppcx4,pf);
```

Brent.mw

```
restart;
with(combinat,permute):
with(plots):
with(plottools):
with(stats):
with(LinearAlgebra):
with(linalg):

hatena:=proc(a,b,c,d)
  if evalf(a)>=evalf(b) then return c;
  else return d;
  end if;
end proc:

mysign:=proc(a,b)
  if evalf(b)<0 then
  return -a;
    else return a;
  end if;
end proc:

fret:=[0,0];
```

```
f:=unapply(x^2-27*x+2,x);
plot(f(x),x=-50..50);

df:=diff(f(x),x);
dfs:=solve(df,x);
f(dfs);
evalf(dfs);evalf(f(dfs));

tol:=1.0e-10;
cgold:=1.0-(sqrt(5.0)-1)/2;
zeps:=1.0e-10;
e:=0.0;d:=0.0;
ax:=-20.0;bx:=0.0;cx:=20.0;

if evalf(ax) < evalf(cx) then
  a:=ax;
else a:=cx;
end if;
if evalf(ax)>evalf(cx) then
  b:=ax;
else
  b:=cx;
end if;
x:=bx; w:=bx; v:=bx;
fw:=f(x);fv:=f(x);fx:=f(x);
 for j from 1 to 100 do
   xm:=0.5*(a+b);
   tol1:=tol*abs(x)+zeps;
   tol2:=2.0*tol1;

  #収束判定
   if evalf(abs(x-xm)) <= evalf(tol2-0.5*(b-a)) then
     xmin:=x;
     fret[1]:=xmin;
     fret[2]:=fx;
     printf("fin");
     break;
   end if;
```

```
  if evalf(abs(e))>evalf(tol1) then
    r:=(x-w)*(fx-fv);
    q:=(x-v)*(fx-fw);
    p:=(x-v)*q-(x-w)*r;
    q:=2.0*(q-r);

    if evalf(q)>0.0 then
      p:=-p;
    end if;

    q:=abs(q);
    etemp:=e;
    e:=d;
    if evalf(abs(p)) >= evalf(abs(0.5*q*etemp)) or
evalf(p)<= evalf(q*(a-x)) or evalf(p) >= evalf(q*(b-x)) then
      e:=hatena(evalf(x),evalf(xm),a-x,b-x);
      d:=cgold*e;
    else
      d:=p/q;
      u:=x+d;
      if evalf((u-a)) < evalf(tol2) or evalf((b-u))<evalf(tol2) then
        d:=mysign(evalf(tol1),evalf(xm-x));
      end if;
    end if;

  else
    e:=hatena(evalf(x),evalf(xm),a-x,b-x);
    d:=cgold*e;
  end if;

  u:=hatena(evalf(abs(d)),evalf(tol1),x+d,x+mysign(evalf(tol1),evalf(d)));
  fu:=f(u);
  if evalf(fu)<=evalf(fx) then
    if evalf(u)>=evalf(x) then
      a:=x;
    else
      b:=x;
    end if;
    v:=w; w:=x; x:=u;
```

```
      fv:=fw; fw:=fx; fx:=fu;
    else
      if evalf(u)<evalf(x) then
        a:=u;
      else
        b:=u;
      end if;
      if evalf(fu)<=evalf(fw) or w=x then
        v:=w; w:=u; fv:=fw; fw:=fu;
      elif evalf(fu) <= evalf(fv) or v=x or v=w then
        v:=u; fv:=fu;
      end if;
    end if;
end do:
xmin:=x;
fret[1]:=xmin;
fret[2]:=fx;
fret;
ppax1:=pointplot([fret[1],fret[2]],
symbol=solidcircle,symbolsize=20,color=red):

pf:=plot(f(h),h=0..30):

display(ppax1,pf);
```

B.2 CG 法の結果

mainrb の実行結果

ターミナル

関数 $f(x,y)=(x-1.0)^2+(y-2.0)^2+\frac{xy(x^2-y^2)}{x^2+y^2}$ を CG 法で最小値探索

```
/Users/matsuzukakeiko/desktop/final/CG% pwd
/Users/matsuzukakeiko/desktop/final/CG
/Users/matsuzukakeiko/desktop/final/CG% ruby main.rb
初期点 [20.0,300.0]
ax,xx,cx=0.0 1.0 2.0
[98.374997014778, 122.42182806961]
[-51.3577433177724, 38.1049068333855]
[-33.2259089432964, 1.72128662515156]
[2.71692253688979, 10.8713673385678]
[4.07093696097699, 5.71462689219479]
[1.32570755157779, 4.33520149704055]
[1.72952235661362, 3.43898910697157]
[1.20687416147849, 3.12446194067733]
[1.31127591277617, 2.88243756777115]
[1.19840904571861, 2.82237166091385]
[1.22037077497141, 2.76094643601671]
[1.1936578395176, 2.74833931174987]
[1.19843544735936, 2.7338983682344]
[1.19190269975882, 2.7308393655481]
[1.19303831268654, 2.72749909745689]
[1.19147657574721, 2.72674892651935]
[1.19175206066786, 2.72595886376993]
[1.19138096782413, 2.72577680998742]
[1.19144699193281, 2.72558966556012]
[1.19135897380039, 2.72554482239661]
[1.19137411445705, 2.7255007581054]
[1.1913550432503, 2.72549217437292]
[1.19135715253226, 2.72547932363407]
[1.19135318379068, 2.72547649814474]
"fin."
0.014771s
```

B.2.1 cg_result.mw

```
restart;
with(plots):
with(plottools):
with(stats):
with(LinearAlgebra):
with(linalg):
with(ListTools):
with(combinat,permute):
with(Statistics):
func1:=(x,y)->(x-1.0)^2+(y-2.0)^2;
func3:=(x,y)->x*y*(x^2-y^2)/(x^2+y^2);
f:=unapply(func1(x,y)+func3(x,y),x,y);
f(1.19135748435146, 2.72548027345249);
dxf:=unapply(diff(f(x,y),x),x);
dyf:=unapply(diff(f(x,y),y),y);\
dr:=300;
cpf:=contourplot(f(x,y),x=-dr..dr,y=-dr..dr):
pf:=plot3d(f(x,y),x=-dr..dr,y=-dr..dr,axes=box):
cpf:=contourplot(f(x,y),x=-dr..dr,y=-dr..dr):
p1:=[20.0,300.0];

ki1:=[98.374997014778, 122.42182806961]:
ki2:=[-51.3577433177724, 38.1049068333855]:
ki3:=[-51.3577433177724, 38.1049068333855]:
ki4:=[2.71692253688979, 10.8713673385678]:
ki5:=[4.07093696097699, 5.71462689219479]:
ki6:=[1.32570755157779, 4.33520149704055]:
ki7:=[1.72952235661362, 3.43898910697157]:
ki8:=[1.20687416147849, 3.12446194067733]:
ki9:=[1.31127591277617, 2.88243756777115]:
ki10:=[1.19840904571861, 2.82237166091385]:
ki11:=[1.22037077497141, 2.76094643601671]:
ki12:=[1.1936578395176, 2.74833931174987]:
ki13:=[1.19843544735936, 2.7338983682344]:
ki14:=[1.19190269975882, 2.7308393655481]:
ki15:=[1.19303831268654, 2.72749909745689]:
ki16:=[1.19147657574721, 2.72674892651935]:
```

```
ki17:=[1.19175206066786, 2.72595886376993]:
ki18:=[1.19138096782413, 2.72577680998742]:
ki19:=[1.19144699193281, 2.72558966556012]:
ki20:=[1.19135897380039, 2.72554482239661]:
ki21:=[1.19137411445705, 2.7255007581054]:
ki22:=[1.1913550432503, 2.72549217437292]:
ki23:=[1.19135715253226, 2.72547932363407]:
ki24:=[1.19135318379068, 2.72547649814474]:

dxf(p1[1]);
dyf(p1[2]);

pp1:=pointplot3d([op(p1),f(op(p1))],symbol=solidbox,
symbolsize=20,color=brown):
pki1:=pointplot3d([op(ki1),f(op(ki1))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki2:=pointplot3d([op(ki2),f(op(ki2))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki3:=pointplot3d([op(ki3),f(op(ki3))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki4:=pointplot3d([op(ki4),f(op(ki4))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki5:=pointplot3d([op(ki5),f(op(ki5))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki6:=pointplot3d([op(ki6),f(op(ki6))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki7:=pointplot3d([op(ki7),f(op(ki7))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki8:=pointplot3d([op(ki8),f(op(ki8))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki9:=pointplot3d([op(ki9),f(op(ki9))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki10:=pointplot3d([op(ki10),f(op(ki10))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki11:=pointplot3d([op(ki11),f(op(ki11))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki12:=pointplot3d([op(ki12),f(op(ki12))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki13:=pointplot3d([op(ki13),f(op(ki13))],symbol=solidcircle,
```

```
symbolsize=15,color=yellow):
pki14:=pointplot3d([op(ki14),f(op(ki14))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki15:=pointplot3d([op(ki15),f(op(ki15))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki16:=pointplot3d([op(ki16),f(op(ki16))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki17:=pointplot3d([op(ki17),f(op(ki17))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki18:=pointplot3d([op(ki18),f(op(ki18))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki19:=pointplot3d([op(ki19),f(op(ki19))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki20:=pointplot3d([op(ki20),f(op(ki20))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki21:=pointplot3d([op(ki21),f(op(ki21))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki22:=pointplot3d([op(ki22),f(op(ki22))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki23:=pointplot3d([op(ki23),f(op(ki23))],symbol=solidcircle,
symbolsize=15,color=yellow):
pki24:=pointplot3d([op(ki24),f(op(ki24))],symbol=solidcircle,
symbolsize=25,color=red):

line1:=[p1,ki1,ki2,ki3,ki4,ki5,ki7,ki8,ki9,ki10,ki11,ki12,
ki13,ki14,ki15,ki16,ki17,ki18,ki19,ki20,ki21,ki22,ki23,ki24]:

pl:=pointplot(line1,connect=true):


display(pf,pp1,pki1,pki2,pki3,pki4,pki5,pki6,pki7,pki8,pki9,pki10,pki11,pki12,
pki13,pki14,pki15,pki16,pki17,pki18,pki19,pki20,pki21,pki22,pki23,pki24);

dr2:=300;
display(cpf,pl,view=[-dr2..dr2,-dr2..dr2]);
```

付 録C　Powell法

C.1　RubyでのPowell法

main.rb，Powell.rb，linmin.rb 以外は CG 法と同じものを使用する．

C.1.1　main.rb

```
t0 = Time.now
require "./Powell.rb"
require 'pp'

#次元
n=2

#初期点 p1[dummy,x,y]
p1=[0.0,20.0,300.0]
ax=-100.0
xx=-40.0
cx=100.0

xfret=[0,0]

#frprmn から
  xi=Array.new(n+1)
  xi=dfunc(p1)
###

#frprmn における収束条件
$frp_ftol=1.0e-10
$frp_eps=1.0e-10

#Brent における収束条件
$Brent_tol=1.0e-10
```

```
$ncom=n
$pcom=Array.new(n+1)
$xicom=Array.new(n+1)

p Powell(p1,xi,n,ax,xx,cx)

t1 = Time.now

print t1-t0,"s","\n"

\begin{verbatim}
\subsection{Powell.rb}
\begin{verbatim}
include Math
require 'pp'
require "linmin.rb"
require "etc.rb"
require "f1dim.rb"

def sqr(a)
  if a==0.0 then return 0.0
  else return a*a
  end
end

def Powell(p1,xi,n,ax,xx,cx)
  itmax=200
  print "p1"; pp p1
  pt=Array.new(n+1)
  ptt=Array.new(n+1)
  xi_a=[]
  xi_b=[]
  xi_a[0]=0.0
  xi_b[0]=0.0
  xit=[0.0,0.0,0.0]
  xi=[[0,0,0],[0,10,15],[0,20,25]]
  #ここで xi 作成終了
  xfret=func(p1)
```

```
ftol= $frp_ftol
eps=$frp_eps
#初期点を保存
for j in 1..n do
  pt[j]=p1[j]
end
for iter in 1..itmax do
  print "p1"; pp p1
  fp=xfret
  print "fp: "; pp fp
  ibig=0
  #関数値の最大限少量を求める変数 del
  del=10e-10
  for i in 1..n do
    for j in 1..n do
      xit[j]=xi[j][i]
    end
    fptt=xfret
    p1,xit,xfret,ax,xx,cx=linmin(p1,xit,n,ax,xx,cx)
    if (fptt-xfret).abs>del then
      del=(fptt-xfret).abs
      ibig=i
    end
  end
  if 2.0*(fp-xfret).abs <= ftol*((fp).abs+(xfret).abs) then
    #収束判定
    p "fin"
    return p1
  end
  if (iter==itmax) then
    print "error"
  end
  for m in 1..n do
    ptt[m]=2.0*p1[m]-pt[m]
    xit[m]=p1[m]-pt[m]
    pt[m]=p1[m]
  end
  fptt=func(ptt)
  if fptt < fp then
```

```
      t=2.0*(fp-2.0*xfret+fptt)*sqr(fp-xfret-del)-del*sqr(fp-fptt)
      if t<0.0 then
        p1,xit,xfret,ax,xx,cx=linmin(p1,xit,n,ax,xx,cx)
        for l in 1..n do
          xi[l][ibig]=xi[l][n]
          xi[l][n]=xit[l]
        end
      end
    end
  end
end
```

C.1.2 linmin.rb

```
require 'pp'
include Math
require "mnbrak.rb"
require "Brent.rb"

def linmin(p1,xi,n,ax,xx,cx)
 # nrfunc=func
  for j in 1..n do
    $pcom[j]=p1[j]
    $xicom[j]=xi[j]
  end
  ax=0.0
  xx=1.0
  ax,xx,cx=mnbrak(ax,xx,cx)
  xfret=Brent(ax,xx,cx)
  xmin = xfret[0]
  z=xfret[1]
    for j in 1..n do
    xi[j]*=xmin
    p1[j]+=xi[j]
    end
  return p1,xi,z,ax,xx,cx  #2/3
end
```

C.2 Powell 法の結果

main.rb の実行結果

ターミナル

関数 $f(x,y) = (x-1.0)^2 + (y-2.0)^2 + \frac{xy(x^2-y^2)}{x^2+y^2}$ を Powell 法で最小値探索

```
/Users/matsuzukakeiko/desktop/final/Powell% pwd
/Users/matsuzukakeiko/desktop/final/Powell
/Users/matsuzukakeiko/desktop/final/Powell% ruby main.rb
p1[0.0, 20.0, 300.0]
p1[0.0, 20.0, 300.0]
fp: 83218.0973451327
p1[0.0, -105.031960699397, 49.5075348957173]
fp: 10190.4193346443
p1[0.0, 0.347759599308915, 0.75095385779246]
fp: 1.81661266254698
p1[0.0, 1.13789146146293, 2.60034981473709]
fp: -1.62841379548685
p1[0.0, 1.19135311864512, 2.72547558795321]
fp: -1.64230430090107
"fin"
[0.0, 1.19135311867942, 2.72547558803349]
0.050876s
```