IBMEC SÃO PAULO

INSPER
Instituto de Ensino e Pesquisa

Leo Satsukawa

TÉCNICA COMPUTACIONAL DE ENXAME DE PARTÍCULAS NA ANÁLISE DE COMPORTAMENTO NO MERCADO FINANCEIRO

São Paulo
2009

Leo Satsukawa

Técnica Computacional de Enxame de Partículas na Análise de Comportamento no Mercado Financeiro

Relatório Parcial de Iniciação Científica.

Orientador:
Prof. Dr. Marco Antonio Leonel Caetano – Insper

São Paulo
2009

Sumário

Capítulo 1 – Introdução

Essa pesquisa tem o intuito de analisar a aplicação da técnica PSO (*Particle Swarm Optimization*) para o estudo do comportamento de investidores dentro do mercado financeiro, sobretudo, bolsa de valores. O algoritmo de PSO se baseia em duas equações que representam a simulação de uma quantidade "n" de partículas. A programação pode envolver problemas complexos. Para tanto, no início foi estudado a programação básica do algoritmo de PSO em duas linguagens computacionais diferentes: VBA e MATLAB. O projeto contemplou em fases posteriores uma programação mais avançada do PSO com o intuito de aprofundar os conhecimentos da linguagem computacional dessa área. Foram analisados dados reais da bolsa de valores (BOVESPA) com a finalidade de observar se há algum movimento de inteligência coletiva.

Caso seja possível observar um determinado comportamento dos investidores no mercado financeiro, pode ser possível entender os movimentos e determinar padrões nas escolhas de compra e venda dos ativos.

Para estudar sobre a técnica PSO foram lidos trabalhos de diversos autores (Kennedy & Eberhart, 1995), (Kendall & Yan Su, 2005).

Iniciou-se com uma programação, na qual era criada uma população com 5 partículas aleatórias. O objetivo era chegar a um determinado ponto mínimo de uma equação. Para alcançar esse objetivo foi utilizada a equação do algoritmo da técnica PSO. O intuito é simular um padrão de busca por melhores retornos dos investidores. Na programação cada partícula atualizava a própria posição a cada iteração de acordo com a melhor posição encontrada até o momento pelo conjunto e a sua posição anterior, ambas ponderadas a uma taxa de cognição. Com a mesma teoria foi feita uma programação com duas populações de 5 partículas se interagindo.

A programação do algoritmo no MATLAB foi um avanço, com as vantagens de possuir ferramentas mais avançadas e apropriadas para o projeto. Da mesma forma que no VBA começou-se com uma única população, contudo, com 10 partículas que tinham como objetivo alcançar um ponto mínimo utilizando o algoritmo de PSO. Essa diferença entre a quantidade de partículas se deve ao fato de que a velocidade da simulação no MATLAB era muito mais rápida e, portanto, mais apto a comportar um número maior de partículas em uma mesma simulação.

A programação desse projeto pôde ser executada tanto na linguagem computacional do VBA quanto na do MATLAB. Pode-se dizer que a única necessidade foi o entendimento do algoritmo, ou seja, as ferramentas e as funções de cada uma das linguagens eram suficientes para fazer a programação

A aquisição dos dados reais sobre a BOVESPA foi feita através do software Economática.

Capítulo 2 – Técnica de Enxame de Partículas

Com a evolução tecnológica dos computadores está sendo possível realizar estudos que necessitam de uma grande quantidade de variáveis, amostras e testes. Frente a isso, pesquisadores começaram a ter interesse em encontrar uma nova forma de inteligência artificial, a inteligência coletiva (*swarm intelligence*). Segundo Graham e Yan Su, a inteligência coletiva originou de estudos sobre populações de seres vivos que possuem um comportamento similar a um enxame. As referências utilizadas comumente como exemplos desse tipo de inteligência são as colônias de formigas, de cupins e os bandos/nuvens de pássaros. Nessas comunidades os indivíduos, separadamente, seguem regras simples, enquanto que quando é observado o enxame em si é possível verificar comportamentos complexos e inteligentes. Uma das razões para tentar encontrar essa forma de inteligência é que o mundo está cada vez mais complexo, existe uma vasta quantidade de informações e de teorias que um único ser humano não é mais capaz de absorvê-los. A inteligência coletiva contrapõe a idéia de que um único indivíduo é um processador de informações totalmente isolado e ressalva o fato que a "inteligência" se forma pelas iterações de agentes inteligentes. Uma das recentes propostas de técnicas de otimização baseada na inteligência coletiva é o PSO.

A técnica PSO ou otimização com enxame de partículas foi originalmente desenvolvido por Kennedy e Eberhart (J. Kennedy, R. C. Eberhart, Particle Swarm Optimization, *Proc. of the IEEE International Conference on Neural Networks*, Vol. 4, PP. 1942-1948, 1995). Basearam-se no modelo de comportamento social dos bandos de aves e tentaram simular graficamente os comportamentos imprevisíveis que esses conjuntos apresentavam, com a finalidade de descobrir um padrão para alguns de seus comportamentos.

A técnica foi desenvolvida segundo algumas definições de seus criadores. Entre elas se destacam:

- **Partícula** – agente que toma decisões a fim de produzir um evento que poderá otimizar a situação da população. Na programação as partículas são representadas por pontos e elas se movimentam de modo que poderá otimizar a função objetivo da população;
- **População** – conjunto de partículas com as mesmas funções objetivo, ou seja, as partículas de uma mesma população têm o mesmo objetivo. Partículas de populações diferentes são representadas por cores diferentes na programação;
- **Otimizar** – encontrar uma situação ótima, ou a melhor situação possível, em um determinado problema;
- **Cognição** – porcentagem de informação que é absorvida pelas partículas em um tempo t para que possam ser levadas em conta na próxima iteração, ou seja, em t+1. A cognição é representada pela letra grega ϕ e até o momento consideramos ser uma variável exógena;
- **Pbest** – são as melhores soluções ou posições encontradas pelas respectivas partículas até o momento;
- **Gbest** – é a melhor solução ou posição encontrada por uma população até o momento.

Cada partícula:

1. Possui uma posição e uma velocidade no instante t;
2. Possui uma ou mais funções objetivo do tipo *f(x(i),y(i))*, na qual *i* é uma partícula;
3. Altera a sua posição de acordo com a sua velocidade quando há uma nova iteração, a fim de encontrar a posição ótima;

Cada população possui:

1. n partículas com as mesmas funções objetivo;
2. A melhor posição até o instante t;
3. 2 taxas de cognição, uma referente à melhor posição de cada partícula até o instante t e outra à melhor posição até o instante t.

t = 1,..., j.

2.1 - Programação do PSO básico

Foi usado um algoritmo básico dividido em 6 passos para a otimização do enxame de partículas:

1. Inicialmente gera-se n partículas com posições e velocidades aleatórias
2. Defini-se a função a ser minimizada;
3. Ajuste do Pbest de cada partícula: compara-se a melhor posição encontrada pela respectiva partícula;
4. Ajuste do Gbest: compara-se a melhor posição encontrada na população;
5. Atualiza-se a velocidade a partir da equação:

   v=v(i)+ϕ1*rand()*(px(i)-x(i)) + ϕ2*rand()*(gx-x(i))

   v=v(i)+ϕ1*rand()*(py(i)-y(i)) + ϕ2*rand()*(gy-y(i)), nas quais:

   - ϕ1 e ϕ2 são taxas de cognição, ou de aprendizagem social ;
   - p() é o valor do eixo cuja posição é a anterior, ou seja o Pbest referente à partícula i;
   - g() é o valor do eixo cuja posição é a melhor possível até o momento, ou seja, o Gbest;
   - rand() é um número aleatório não inteiro que varia de 0 a 1.

6. Volte para o Passo 2, repetir o Passa 6 enquanto um critério pré-estabelecido não é alcançado.

Esse algoritmo foi programado no VBA e no MATLAB, fazendo os ajustes necessários em cada linguagem para que seja possível executá-lo.

As taxas de cognições determinam a velocidade que será encontrada a solução ótima, ou a melhor solução possível, de uma população. Quanto maior forem elas menor será o tempo necessário para as partículas chegarem à posição desejada, ou uma solução muito próxima dela.

Capítulo 3 – Aplicação

Neste capítulo será bordado como as programações foram feitas em diferentes cenários e linguagens, no caso desta pesquisa, no VBA e no MATLAB. Os cenários propostos foram os de simulações com uma única população, duas populações independentes, duas populações interagindo entre si, diferentes taxas de cognições e restrições das trajetórias. Cenários como a de duas populações interagindo entre si e o de restrições das trajetórias foram aplicadas somente no MATLAB devido a presença de ferramentas mais eficientes nesse programa do que no VBA.

3.1 - O Potencial do VBA

A programação no VBA foi feita em um *Userform*, na qual há um *CommandButton* que inicia o programa, como na Figura 3.1 .

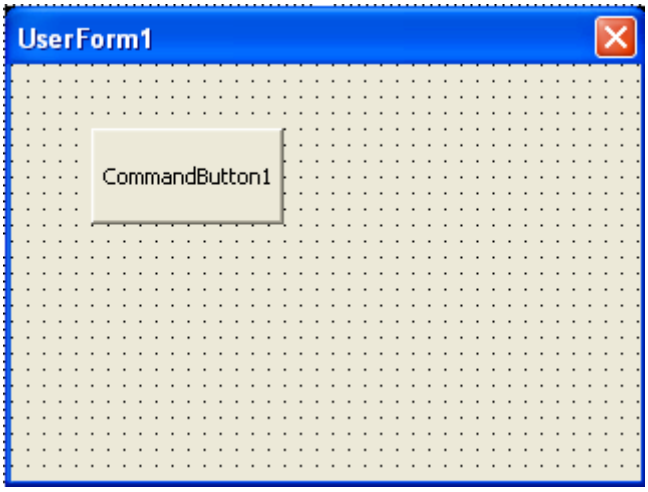


Figura 3.1 – *Userform utilizado para a programação do PSO.*

População Única

Nesse programa gera-se aleatoriamente uma população com 5 partículas que buscam o mínimo de uma função f(x, y), atualizando a sua posição a cada iteração de acordo com a velocidade e a sua posição anterior, ponderadas a uma taxa de cognição ϕ_i, na qual i= 1, 2.

O programa inicial foi baseado nas seguintes partes principais dos algoritmos a seguir:

Primeiro foram geradas as variáveis necessárias para realizar o programa, incluindo vetores e parâmetros relacionados ao PSO. Após essas escolhas, foram incluídas as linhas a seguir,

```
Range(Cells(1, 1), Cells(100, 100)).delete '-----------------------------------------------(1)

Range(Cells(1, 1), Cells(100, 100)).Interior.Color = vbWhite

fi1 = 0.5 '-------------------------------------------------------------------------------(2)
fi2 = 0.5


fbest = 100000 '--------------------------------------------------------------------------(3)

Set W = Worksheets("Plan1") '-------------------------------------------------------------(4)

W.Shapes.AddLine(40, 35, 40, 45).Select '-------------------------------------------------(5)
W.Shapes.AddLine(35, 40, 45, 40).Select

Randomize  '------------------------------------------------------------------------------(6)
```

A primeira linha, (1), tem a finalidade de limpar a planilha, a fim de começar uma nova representação da otimização por enxame de partículas. A segunda, logo abaixo, faz com que a planilha seja preenchida com um fundo branco, para que seja possível uma melhor visualização dos movimentos.

Foram definidas as cognições que serão usadas na equação em (2). Note que há duas taxas, *fi1 e fi2*, ambas não precisam ser necessariamente iguais. Elas são representações das taxas de cognições das populações

Foi escolhido um valor inicial para o *fbes*t, que é uma variável que representa a melhor posição até o momento, em (3). Ele possui um valor alto por uma razão que será discutida mais a frente, assim como os valores de x e y que o compõe.

Em (4) é atribuído um nome qualquer, no caso seria *W*, para a planilha que será observada as partículas e seus movimentos a fim de facilitar na hora de usar uma função que seja necessária escolher uma determinada planilha.

A função *W.Shapes* fará com que na planilha *W*, nomeada em (4), seja inserida alguma figura geométrica. Em (5), usa-se dois *AddLine*, que fará com que sejam adicionadas duas linhas com as coordenadas *(40, 35, 40, 45)* e *(35, 40, 45, 40)*, formando uma cruz com centro em *x(i)=40* e *y(i)=40*. Esse *Shape* servirá para representar o ponto ótimo da a equação otimizar sugerido para esta simulação.

Rnd é uma função que gera números entre 0 a 1, contudo como não é totalmente aleatório foi usada um outra função, a *Randomize*, para tornar isso possível em (5).

```
For i = 1 To 5

    x(i) = (100 * Rnd) '-------------------------------------------------------------------(7)
    y(i) = (100 * Rnd)

    vx(i) = (100 * Rnd) '------------------------------------------------------------------(8)
    vy(i) = (100 * Rnd)

    f(i) = (x(i) - 40) ^ 2 + (y(i) - 40) ^ 2 '---------------------------------------------(9)

    With W.Shapes.AddShape(msoShapeOval, x(i), y(i), 2, 2) '-----------------------------(10)
    .Fill.ForeColor.RGB = RGB(0, 0, 255)
    End With

        xbest(i) = x(i) '--------------------------------------------------------------------(11)
        ybest(i) = y(i)

    If f(i) < fbest Then '---------------------------------------------------------------(12)
        fbest = f(i)
        gx = x(i)
        gy = y(i)
    End If

Next i
```

Foi criado uma função *for i=n* para poder gerar n partículas, que no caso é 5. Sabendo que a posição de uma partícula é dada pela função *f(x(i), y(i))*, na qual *i= 1, 2, 3, 4, 5*, foi necessário gerar *x(i)* e *y(i)* de modo aleatório por (7). Para isso foi utilizado a função *Rnd*, que junto com a função *Randomize* citada anteriormente fará com que seja possível gerar números de 0 a 1 de modo totalmente aleatório. A *Rnd* foi multiplicada por 100, pela conveniência na hora de visualizar os movimentos das partículas, caso contrário teríamos que tentar visualizar a simulação em um quadrante muito pequeno. De maneira análoga foi gerada uma velocidade inicial para cada partícula, como em (8), na qual *vx(i)* e *vy(i)* representam a velocidade da partícula *i* nos eixos x e y, respectivamente.

Em (9) foi definida uma função *f(x(i), y(i)) = (x(i)-40) ^ 2 +(y(i)-40) ^ 2* para a população dessa simulação, que no caso será uma minimização. Essa equação terá uma

solução de mínimo num ponto ótimo com as coordenadas iguais a *x(i)=40* e *y(i)=40*, de modo que *f(x,y)=0*.

Para que seja possível observar as partículas na planilha do Excel foi usado a função *msoShabeOval* como em (10),ou seja, serão representadas por formas circulares, preenchidas com a cor azul conforme *.Fill.ForeColor.RGB = RGB(0, 0, 255)* e cada uma estará localizada na posição *f(x(i),y(i))*.

Como não há nenhuma iteração anterior até essa fase, foi definido que o *xbest* e o *ybest* de cada partícula são iguais ao *x(i)* e ao *y(i)*, respectivamente, como em (11). Na teoria é o mesmo afirmar que o pbest de cada partícula é a sua própria posição inicial.

Foi definido um *fbest=1000000*, que é suficientemente grande, com o objetivo de que uma das posições iniciais das partículas geradas seja definida como a melhor posição até o momento, ou seja, *fbest* (na teoria seria o Gbest) como em (12).

```
k = 1 '-------------------------------------------------------------------------------(13)
erro = 100000 '-----------------------------------------------------------------------(14)
```

k, em (13), é um contador que limitará o número de iterações, seu algoritmo será explicado nas linhas abaixo.

A linha (14) representa o *erro*, que será a distância entre as partículas e a solução ótima e uma segunda maneira de limitar o número de iterações. Quando uma partícula alcança uma posição muito próxima da solução ótima (0,01 na programação abaixo) o programa pára de executar imediatamente. Isso ocorre devido à função *Do While erro > 0.01* em (15), ou seja, o programa continuará com as iterações até que o *erro* seja menor ou igual 0,01. Para iniciar a programação foi suposto que o *erro* fosse alto (igual a 100000), caso seja muito baixo (menor do que 0,01 segundo (15)) o programa iria finalizar imediatamente, antes mesmo de ocorrer a próxima iteração.

```
Do While erro > 0.01 '-------------------------------------------------------------------(15)

    If k > 100 Then '-------------------------------------------------------------------(16)
        erro = 0
    Else

        For i = 1 To 5 '-------------------------------------------------------------------(17)

            vvx = vx(i) + (fi1 * Rnd) * (xbest(i) - x(i)) + (fi2 * Rnd) * (gx - x(i)) '---(18)
            vvy = vy(i) + (fi1 * Rnd) * (ybest(i) - y(i)) + (fi2 * Rnd) * (gy - y(i))

            xx = vvx + x(i) '-----------------------------------------------------------(19)
            yy = vvy + y(i)

            f(i) = (x(i) - 40) ^ 2 + (y(i) - 40) ^ 2 '-----------------------------------(20)

            xbest(i) = x(i) '-----------------------------------------------------------(21)
            ybest(i) = y(i)


            If f(i) < fbest Then '-------------------------------------------------------(22)
                fbest = f(i)
                gx = x(i)
                gy = y(i)
            End If

            x(i) = xx '-------------------------------------------------------------------(23)
            y(i) = yy
            vx(i) = vvx
            vy(i) = vvy

        Next i

    End If
```

Observe que em (16) há uma função de condição *if* que se *k*, ou seja, o número de iterações, for maior do que 100 faz também com que o programa pare a execução, pois fará com que o *erro* seja igual a 0 e segundo (15) não será possível prosseguir para uma nova iteração. Essa condição foi adicionada, pois nem sempre será possível encontrar a solução ótima em um curto prazo de tempo. Além disso, caso as taxas de cognições sejam muito baixas a simulação demandará um tempo enorme para que seja concluída, ou seja, até encontrar a solução ótima.

A função *For* em (17) tem um contador *i* que vai de 1 a 5 para poder atualizar a velocidade e a posição das 5 partículas, a fim de que seja possível visualizar uma nova iteração. Dentro dessa função vemos os algoritmos (18), (19), (20), (21) e (22), que são as atualizações da velocidade, das posições nos eixos, da posição na função, do Pbest de cada partícula e do Gbest da população, respectivamente. Note que em (19) a nova posição da partícula *i* foi inserida em *xx(i)* e *yy(i)* ao invés de simplesmente *x(i)* e *y(i)*, pois *xx(i)* e *yy(i)* são as posições da partícula, nos eixos, da próxima iteração, assim como a velocidade em (20).

Em (23) o algoritmo tem a finalidade de reescrever os valores de *xx(i)* e *yy(i)* em *x(i)* e *y(i)*, além de *vvx(i)* e *vvy(i)* em *vx(i)* e *vy(i)*, respectivamente, para que seja possível efetuar a atualização das velocidades das partículas na próxima iteração.

```
    For Each pto In ActiveSheet.Shapes '-------------------------------------------------(24)
        pto.delete
    Next pto

    W.Shapes.AddLine(40, 35, 40, 45).Select
    W.Shapes.AddLine(35, 40, 45, 40).Select

    For i = 1 To 5  '-------------------------------------------------------------------(25)
        With W.Shapes.AddShape(msoShapeOval, x(i), y(i), 2, 2)
        .Fill.ForeColor.RGB = RGB(0, 0, 255)
        End With
    Next i

    Application.Wait (Now + TimeValue("0:00:01")) '-------------------------------------(26)

    k = k + 1 '-------------------------------------------------------------------------(27)
    erro = Abs(fbest - 0)

Loop '-----------------------------------------------------------------------------------(28)
End

End Sub '--------------------------------------------------------------------------------(29)
```

(24) é uma função *For* que apaga somente as partículas da planilha, para que em (26) sejam reescritas de modo que seja possível observar a troca de posições delas.

A função *For*, em (25), insere as partículas na planilha *W*, com suas respectivas posições, conforme a função da sua população. Observe que essa função possui um algoritmo idêntico a (10), ou seja, ocorreu uma reprodução. Esse algoritmo é repetido, pois o algoritmo em (24) fez com que tudo seja apagado da planilha. Para podermos observar as iterações seguintes será necessário inserir novamente as partículas, com suas posições atualizadas, na planilha, como em feito (25).

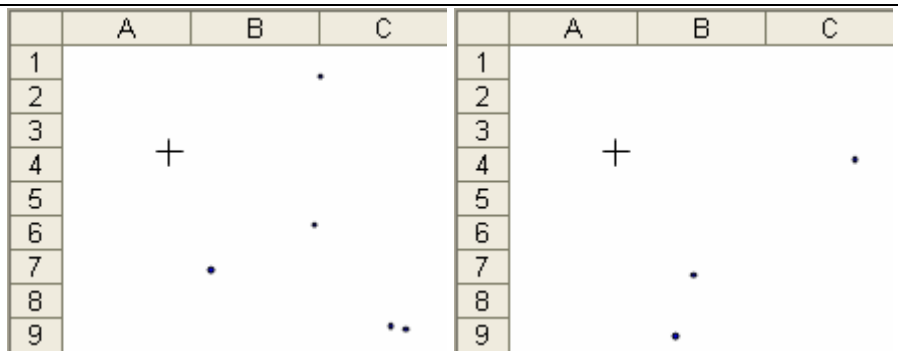
A função em (26) tem o objetivo de pausar o programa depois que é feito as atualizações que estão em (17), isso é necessário para poder observar o movimento das partículas. Caso não tivesse sido criado, a execução do programa iria finalizar instantaneamente.

Em (27) são atualizados o contador *k*, de maneira que a próxima iteração será a *k*+1 ésima iteração, e o *erro*.

É realizado um *Loop* em (28) para que essa parte da programação continue sendo executada enquanto a condição *Do While* é válida, ou seja, enquanto *erro>0.01*.

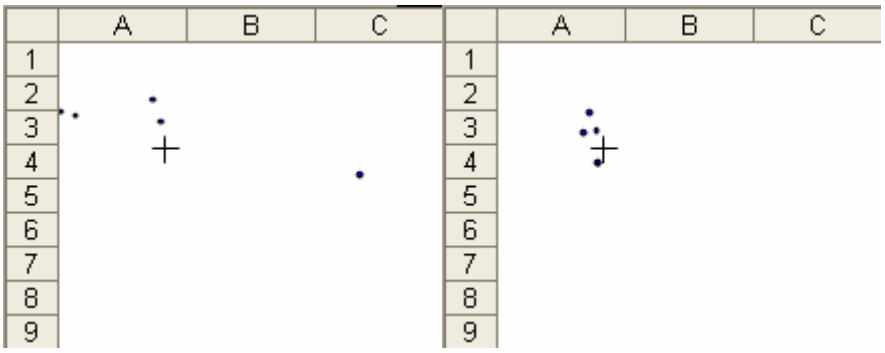
O *EndSub*, em (29), sinaliza o fim da programação.

Quando o programa é iniciado, a partir do *CommandButton*, são geradas todas as partículas em k = 1 e é realizado no máximo 100 iterações (k = 100), como é exemplificado na Figura 3.2. É necessário supor que as colunas da planilha do Excel representam o eixo y e as linhas o eixo x.



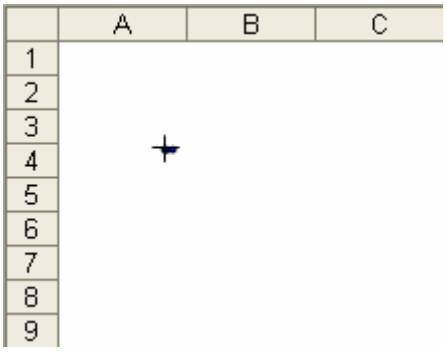
(k=1)

(k=5)



(k=10)

(k=25)



(k=35)

Figura 3.2 – *Representação gráfica da simulação do PSO de população única, para um fi1=fi2=0,5 no VBA.*

Conforme a função *f(x(i),y(i))*, a população na Figura 3.2 convergiu em valores muito próximos a x = 40 e y = 40. Note que na iteração *k=35* foi finalizada a simulação, ou seja, pelo menos uma das partículas convergiu no ponto ótimo, representado pela cruz programada em (5), com um erro máximo de 0,01, segundo (15).

Para observar detalhadamente esta programação no VBA vide o Anexo 1.

Duas Populações Independentes

Foi feita uma programação com duas populações, ao invés de uma única como na anterior, independentes, ou seja, uma não influenciará na outra. A solução é chegar a uma posição ótima, ou muito próxima dela, que é um ponto de mínimo, na qual varia de acordo com as equações de cada população.

Da mesma forma que na primeira programação, foi criado um *userform* com um *CommandButton* que inicia o programa.

Esta simulação segue a mesma linha de programação que a de uma única população, contudo, nas linhas (2) a (17), (19) e (20) foi feita uma duplicação dos algoritmos, ajustando com a respectiva população de maneira a não entrar em conflito com o programa. Por exemplo, em (2), onde na programação de uma população havia somente *fi1* e *fi2*, foram adicionadas mais duas taxas de cognição, contudo, relacionados à segunda população. Para não confundir as variáveis, chamou-se de *fi11* e *fi21*, as taxas de cognição referentes à primeira população e de *fi12* e *fi22* as taxas da segunda população.

```
Range(Cells(1, 1), Cells(100, 100)).delete

Range(Cells(1, 1), Cells(100, 100)).Interior.Color = vbWhite

ActiveSheet.Shapes.AddLine(10, 5, 10, 15).Select '-------------------------------------------------(1)
ActiveSheet.Shapes.AddLine(5, 10, 15, 10).Select

fi11 = 0.5 '-----------------------------------------------------------------------------------------(2)
fi21 = 0.5
fi12 = 0.5
fi22 = 0.5

fbest = 1000000 '------------------------------------------------------------------------------------(3)
f2best = 1000000

Set W = Worksheets("Plan1")

Randomize
```

Foi adicionado em (1) duas funções que criam uma reta cada uma, uma horizontal e outra vertical, para formar uma cruz e facilitar a identificação do ponto ótimo desse problema.

Em (6) foram criadas duas equações, uma para cada população, *f(i) = (x(i)-10)^2 + (y(i)-10)^2* e *f2(i) = (x2(i) - 10)^2 + (y(i) - 10)^2*. As duas equações possuem o mesmo formato, pois, o intuito dessa programação é analisar o potencial do VBA na simulação de duas populações. Portanto, ambas as populações possuirão o mesmo ponto ótimo, que será a solução desse problema de minimização.

```
For i = 1 To 5

    x(i) = (100 * Rnd) '---------------------------------------------------------------------------(4)
    y(i) = (100 * Rnd)
    x2(i) = (100 * Rnd)
    y2(i) = (100 * Rnd)

    vx(i) = (100 * Rnd) '--------------------------------------------------------------------------(5)
    vy(i) = (100 * Rnd)
    vx2(i) = (100 * Rnd)
    vy2(i) = (100 * Rnd)

    f(i) = (x(i) - 10) ^ 2 + (y(i) - 10) ^ 2 '-----------------------------------------------------(6)
    f2(i) = (x2(i) - 10) ^ 2 + (y2(i) - 10) ^ 2

    With W.Shapes.AddShape(msoShapeOval, x(i), y(i), 2, 2) '---------------------------------------(7)
    .Fill.ForeColor.RGB = RGB(0, 0, 255)
    End With

    With W.Shapes.AddShape(msoShapeOval, x2(i), y2(i), 2, 2)
    .Fill.ForeColor.RGB = RGB(255, 0, 255)
    .Line.ForeColor.SchemeColor = 14
    End With

        xbest(i) = x(i) '------------------------------------------------------------------------(8)
        ybest(i) = y(i)

        x2best(i) = x2(i)
        y2best(i) = y2(i)
```

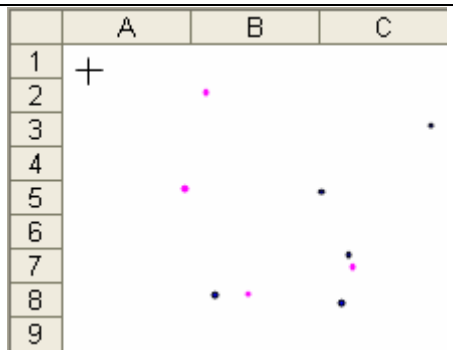
Observe que em (7), no segundo *AddShape*, as diferenças com o primeiro é a posição das partículas nos eixos x e y , os valores em *Fill.ForeColor = RGB* e a adição de um *Line.ForeColor.SchemeColor*. Na função *Fill.ForeColo = RGB*, os valores que o preenchem correspondem à proporção das cores vermelha, azul e amarela, e em (7) as partículas da

primeira população serão representadas pela cor azul, enquanto que as da segunda, pela cor rosa. A função *Line.ForeColor.SchemeColor* altera a cor do contorno de um *AddShape*, na qual em (7) foi escolhida a cor de número 14, ou seja, rosa.

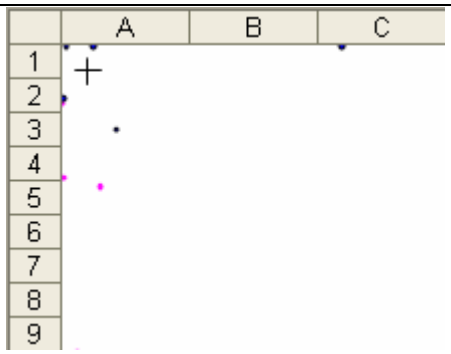
```
        xbest(i) = x(i) '-------------------------------------------------------------------------(8)
        ybest(i) = y(i)

        x2best(i) = x2(i)
        y2best(i) = y2(i)

    If f(i) < fbest Then '--------------------------------------------------------------------------(9)
        fbest = f(i)
        gx = x(i)
        gy = y(i)
    End If
    If f2(i) < f2best Then
        f2best = f2(i)
        gx2 = x2(i)
        gy2 = y2(i)
    End If

Next i

k = 1

erro = 100000 '-----------------------------------------------------------------------------------(10)
erro2 = 100000

Do While erro > 0.01 And erro2 > 0.01 '-----------------------------------------------------------(11)

    If k > 100 Then
        erro = 0
        erro2 = 0
    Else

        For i = 1 To 5

            vvx = vx(i) + (fi11 * Rnd()) * (xbest(i) - x(i)) + (fi21 * Rnd()) * (gx - x(i)) '-----(12)
            vvy = vy(i) + (fi11 * Rnd()) * (ybest(i) - y(i)) + (fi21 * Rnd()) * (gy - y(i))
            vvx2 = vx2(i) + (fi12 * Rnd()) * (x2best(i) - x2(i)) + (fi22 * Rnd()) * (gx2 - x2(i))
            vvy2 = vy2(i) + (fi12 * Rnd()) * (y2best(i) - y2(i)) + (fi22 * Rnd()) * (gy2 - y2(i))

            xx = vvx + x(i) '-------------------------------------------------------------------(13)
            yy = vvy + y(i)
            xx2 = vvx2 + x2(i)
            yy2 = vvy2 + y2(i)

            f(i) = (x(i) - 10) ^ 2 + (y(i) - 10) ^ 2 '-------------------------------------------(14)
            f2(i) = (x2(i) - 10) ^ 2 + (y2(i) - 10) ^ 2

            xbest(i) = x(i) '-------------------------------------------------------------------(15)
            ybest(i) = y(i)
            x2best(i) = x2(i)
            y2best(i) = y2(i)
```

```
            If f(i) < fbest Then '-------------------------------------------------------------------(16)
                fbest = f(i)
                gx = x(i)
                gy = y(i)
            End If
            If f2(i) < f2best Then
                f2best = f2(i)
                gx2 = x2(i)
                gy2 = y2(i)
            End If

            x(i) = xx '-------------------------------------------------------------------------------(17)
            y(i) = yy
            x2(i) = xx2
            y2(i) = yy2

            vx(i) = vvx
            vy(i) = vvy
            vx2(i) = vvx2
            vy2(i) = vvy2
        Next i

    End If

    For Each pto In ActiveSheet.Shapes
        pto.delete
    Next pto

    For i = 1 To 5 '-------------------------------------------------------------------------------(18)
        With W.Shapes.AddShape(msoShapeOval, x(i), y(i), 2, 2)
        .Fill.ForeColor.RGB = RGB(0, 0, 255)
        End With

        With W.Shapes.AddShape(msoShapeOval, x2(i), y2(i), 2, 2)
        .Fill.ForeColor.RGB = RGB(255, 0, 255)
        .Line.ForeColor.SchemeColor = 14
        End With
    Next i

    ActiveSheet.Shapes.AddLine(10, 5, 10, 15).Select '----------------------------------------(19)
    ActiveSheet.Shapes.AddLine(5, 10, 15, 10).Select

    Application.Wait (Now + TimeValue("0:00:01"))

    k = k + 1
    erro = Abs(fbest - 0) '-------------------------------------------------------------------------(20)
    erro2 = Abs(f2best - 0)

Loop
End

End Sub
```

Abaixo as representações gráficas dessa simulação:



(k=1)



(k=5)

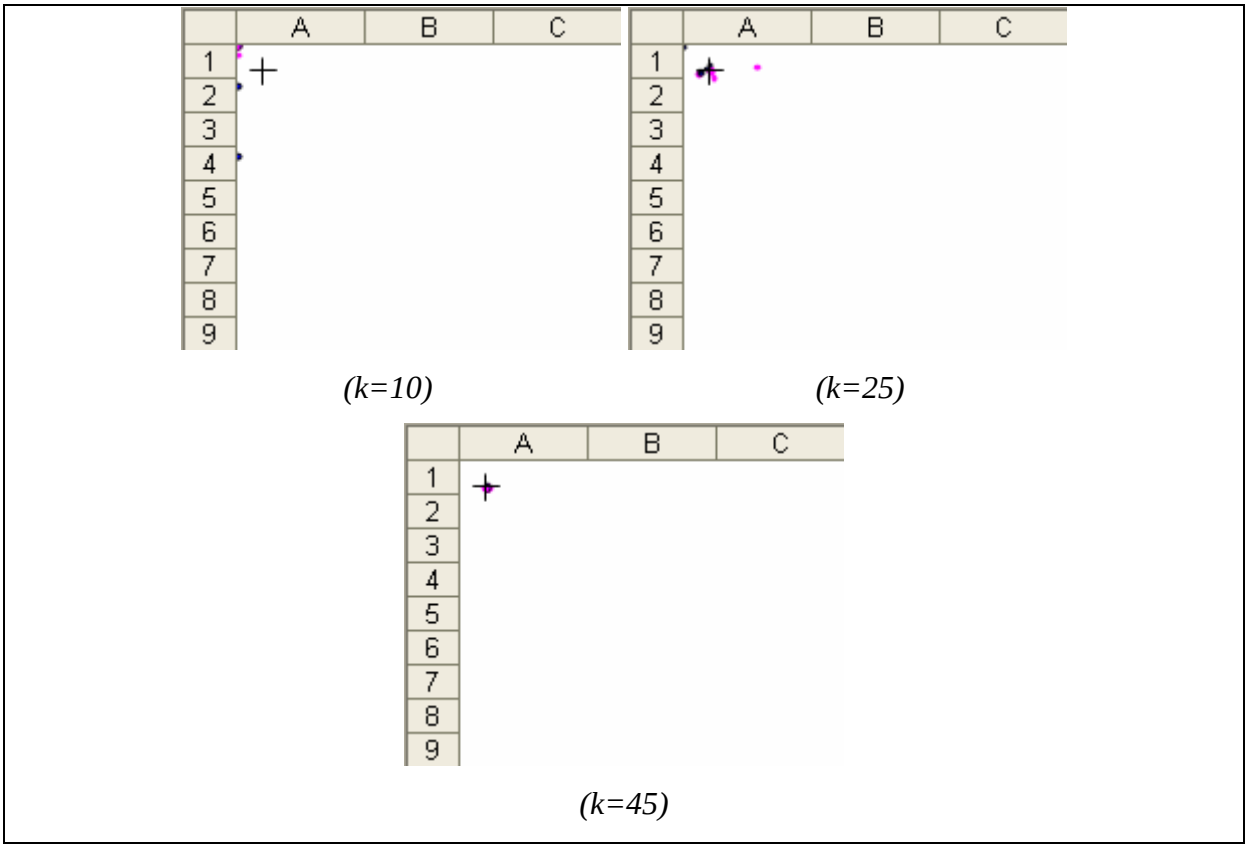


Figura 3.3 – *Representação gráfica da simulação do PSO de duas populações independentes entre si, para um fi11=fi21=fi12=fi22=0,5 no VBA.*

Na Figura 3.3, ambas as populações convergiram em volta da solução ótima, com um erro máximo de 0,01. Como possuíam a mesma solução, faz sentido o fato de que as suas partículas estejam uma próxima da outra.

Para observar detalhadamente esta programação no VBA vide o Anexo 2.

3.2 – Melhorando com o MATLAB

O MATLAB consegue realizar as iterações da programação de PSO a uma velocidade muito maior do que no VBA. Isso faz com que a análise dos movimentos das partículas nesse programa seja mais prática.

Como a linguagem computacional do MATLAB é muito parecida com a do VBA em muitos aspectos, foram necessárias poucas mudanças na programação de simulação do PSO.

As programações de uma simulação do PSO no MATLAB foram todas feitas em um arquivo M-file. Para que

População Única

Como foi feito no VBA, essa programação aborda uma única população com partículas tentando encontrar a solução ótima. No entanto, como o MATLAB possui ferramentas e desempenho mais adequado para esse tipo de programação elevou-se o número de partículas de 5 para 10.

Essa programação foi baseada nos seguintes algoritmos:

```
1 -  tic %-------------------------------------------------------------(1)
2
3 -  clear all %-------------------------------------------------------(2)
4
5 -  fi1=0.5; %--------------------------------------------------------(3)
6 -  fi2=0.5;
7
8 -  fbest=100000; %---------------------------------------------------(4)
```

A função *tic*, em (1), e a *toc*, em (33), foram adicionadas para saber o tempo que simulação levou até ela se encerrar, seja pela população encontrar uma solução muito próxima daquela que seria a melhor possível, ou pelo fato de ter excedido o número máximo de iterações. Abaixo, na Figura 3.4, a visualização dessa função.

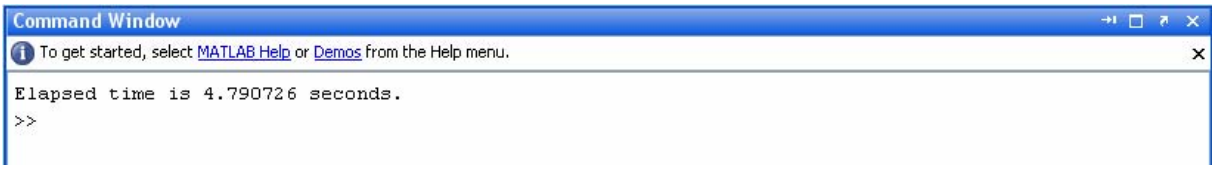


Figura 3.4 – *Command Window com o tempo gasto para a simulação*

Observe que na Figura 3.4 o tempo gasto para que a população encontre a solução ótima foi de 4,79, aproximadamente. Uma simulação que no VBA iria demorar muito mais.

Clear all, em (2), tem a função de apagar todas as informações para que seja possível realizar uma nova simulação.

Em (3) foram adotadas as taxas de cognições *fi1=0.5* e *fi2=0.5*, sendo que *fi1* está relacionado com as informações passadas sobre a partícula, ou seja, em iterações anteriores, e *fi2* com o Gbest.

Como no VBA foi escolhido um valor alto para o *fbest* em (4).

```
 9
10 -  for i=1:10 %-------------------------------------------------------(5)
11
12 -      x(i)=(100*rand); %-----------------------------------------------(6)
13 -      y(i)=(100*rand);
14 -      vx(i)=(100*rand);
15 -      vy(i)=(100*rand);
16
17 -      f(i)=(x(i)-40)^2+(y(i)-40)^2; %----------------------------------(7)
18
19 -      px(i)=x(i); %----------------------------------------------------(8)
20 -      py(i)=y(i);
21
22 -      if f(i)<fbest %--------------------------------------------------(9)
23 -          fbest=f(i);
24 -          gx=x(i);
25 -          gy=y(i);
26 -      end
27
28 -  end
```

A função *For* em (5) tem o objetivo de gerar as posições e velocidades de cada partícula.

Em (6) são geradas a posição da partícula *i* no eixo x e no y e uma velocidade para cada um dos eixos.

Em (7) foi escolhida uma função com solução do tipo minimização, *f(i)=(x(i)-40)^2+(y(i)-40)^2*. Note que essa função então terá uma solução ótima de minimização quando *x(i)=40* e *y(i)=40*, na qual *f(x(i)=40,y(i)=40)=0*.

Como não houve nenhuma iteração, foi suposto que o Pbest de cada partícula fosse a posição atual, como em (8).

Em (9) a função *For* está procurando o Gbest entre as 10 partículas geradas.

```
29
30 -  k=1; %-----------------------------------------------------------(10)
31
32 -  erro=100000; %-------------------------------------------------------(11)
```

Foi definido em *k=1*, em (10), de forma que essa seja a primeira iteração.

Foi suposto em *erro* alto, em (11), nessa primeira iteração para que se possa dar continuidade à simulação.

```
33
34 -  while erro>0.01 %-----------------------------------------------------(12)
35
36 -      if k>100 %-------------------------------------------------------(13)
37 -          erro=0;
38 -      else
39
40 -          for i=1:10 %-----------------------------------------------(14)
41
42 -              vvx=vx(i)+(fi1*rand)*(px(i)-x(i))+(fi2*rand)*(gx-x(i)); %(15)
43 -              vvy=vy(i)+(fi1*rand)*(py(i)-y(i))+(fi2*rand)*(gy-y(i));
44
45 -              xx=x(i)+vvx; %---------------------------------------(16)
46 -              yy=y(i)+vvy;
47
48 -              f(i)=(x(i)-40)^2+(y(i)-40)^2; %-----------------------(17)
49
50 -              px(i)=x(i); %-----------------------------------------(18)
51 -              py(i)=y(i);
52
53 -              if f(i)<fbest %---------------------------------------(19)
54 -                  fbest=f(i);
55 -                  gx=x(i);
56 -                  gy=y(i);
57 -              end
```

As funções *While*, em (12), e *If*, em (13), são condições para que a programação dê prosseguimento à simulação. Caso o erro, a distância entre o Gbest, ou *fbest*, e a solução ótima seja menor do que 0,01 a programação irá parar automaticamente. Mas, se o número de iterações, *k*, for maior do que 100, o mesmo ocorrerá. A razão disso foi explicada anteriormente em **3.1**.

Sob essas condições ((12) e (13)) realiza-se a atualização de cada partícula, em (14), conforme a equação de PSO. Dentro de (14) foi inserido os algoritmos de atualização da

partícula *i* para a velocidade, nos eixos x e y ((15)), as posições em cada eixo ((16)), a posição na função *f(x(i),y(i)* ((17)), do Pbest ((18)) e doGbest ((19)). Foi inserido a nova posição da partícula *i* inicialmente em *xx* e em *yy*, para depois em *x(i)* e *y(i)* ((20)), como explicado em **3.1**, assim como a velocidade, inicialmente computada em *vvx* e *vvy*, para depois em *vx(i)* e *vy(i)* ((21)).

```
58
59 -              x(i)=xx; %-----------------------------------------------(20)
60 -              y(i)=yy;
61
62 -              vx(i)=vvx; %---------------------------------------------(21)
63 -              vy(i)=vvy;
64
65 -          end
66
67 -      plot(x,y,'.', 'markersize',20) %-------------------------------(22)
68 -      axis([-100 100 -100 100]) %------------------------------------(23)
69 -      hold on %------------------------------------------------------(24)
70 -      plot(gx,gy,'+r', 'markersize', 10) %---------------------------(25)
71 -      axis([-100 100 -100 100]) %------------------------------------(26)
72 -      pause(0.1) %---------------------------------------------------(27)
73 -      grid %---------------------------------------------------------(28)
74 -      hold off %-----------------------------------------------------(29)
75
76 -      erro=abs(fbest-0); %-------------------------------------------(30)
77 -      k=k+1; %-------------------------------------------------------(31)
78
79 -      end
80
81 -      pause %--------------------------------------------------------(32)
82
83 -  end
84
85 -  toc %----------------------------------------------------------------(33)
```

A linha (22) é uma função que fará com que as partículas sejam plotadas no gráfico do MATLAB, com as suas respectivas posições em cada eixo. Elas serão representadas por um ponto conforme *'.'* e como nenhuma cor foi especificada pela função, será azul (padrão do MATLAB).*'marksize'* indica que queremos alterar o tamanho padrão desses pontos e *20* foi escolhido como tamanho desejado para analisarmos a simulação..

A linha (23) representa a dimensão do gráfico usado em (22). Como no caso acima, em que *axis ([-100 100 -100 100])*, o eixo x vai de -100 a 100, os primeiros dois números desta função, e o eixo y de -100 a 100, os dois últimos números desta função.

As linhas (24) e (29) correspondem à função *hold*, ela é iniciada em (24) por um *hold on* e terminada em (29) pelo *hold off*. Em conjunto com a função (32), *pause(0.1)*, faz com que a programação seja congelada durante o tempo definido pela função *pause*, ou seja, 0,1 segundos. Essa função é necessária

A linha (25) faz com que uma cruz, do tipo "+", seja inserida no gráfico da simulação, de modo que represente o Gbest quando escrevemos *'plot(gx,gy,'+r', 'marksize',10)'*. Esta cruz terá uma cor vermelha, conforme está em *'+r'* e terá um tamanho 10, conforme *'marksize',10*.

Em (27) há uma função que irá pausar a iteração por 0.1 segundo, para que seja possível visualizar a simulação sem que ela gaste muito tempo, como no VBA em **3.1** que cada iteração demorava 1 segundo.

A função *grid* em (28) tem o objetivo de mostrar a representação gráfica no final da simulação a partir de um gráfico matricial, como no exemplo abaixo:

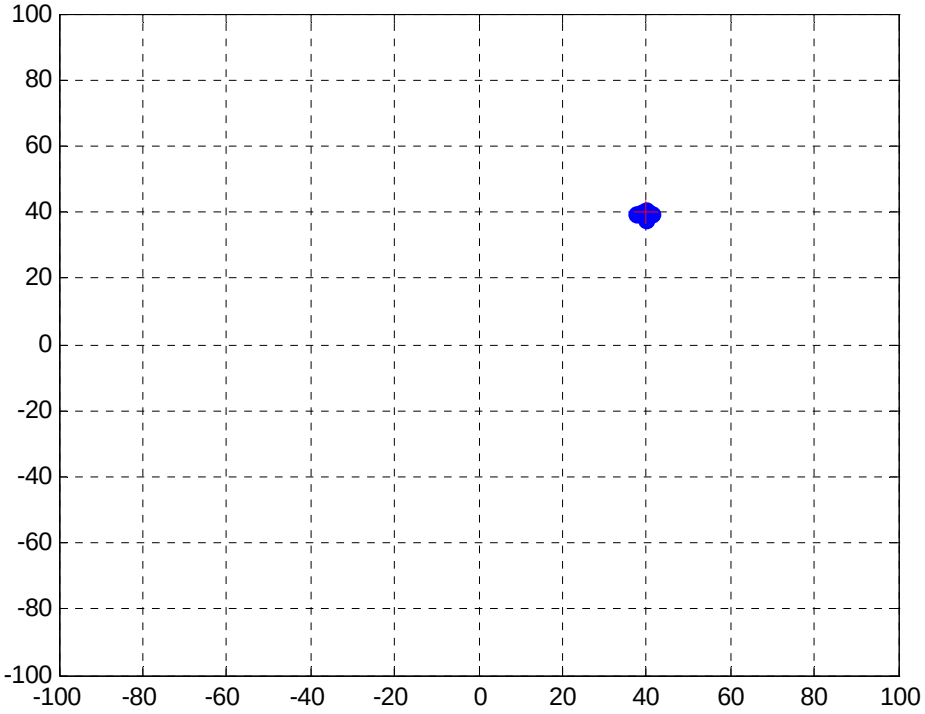


Figura 3.5 – *Exemplo de gráfico matricial, usando a função grid.*

As linhas (30) e (31) atualizam o *erro* e o *k*, respectivamente, para a próxima iteração.

A função *pause*, em (32), não possui um tempo específico de pausa, isso faz com que a simulação só consiga avançar para a próxima iteração quando apertamos a tecla ENTER. Ela foi inserida com a finalidade de podermos analisar os movimentos das partículas passo a passo, ou seja, de uma iteração a outra.

A programação é iniciada a partir da ferramenta *run* do MATLAB, o que fará com que apareça um gráfico que representará a simulação de PSO descrita acima, como na Figura 3.6 abaixo:

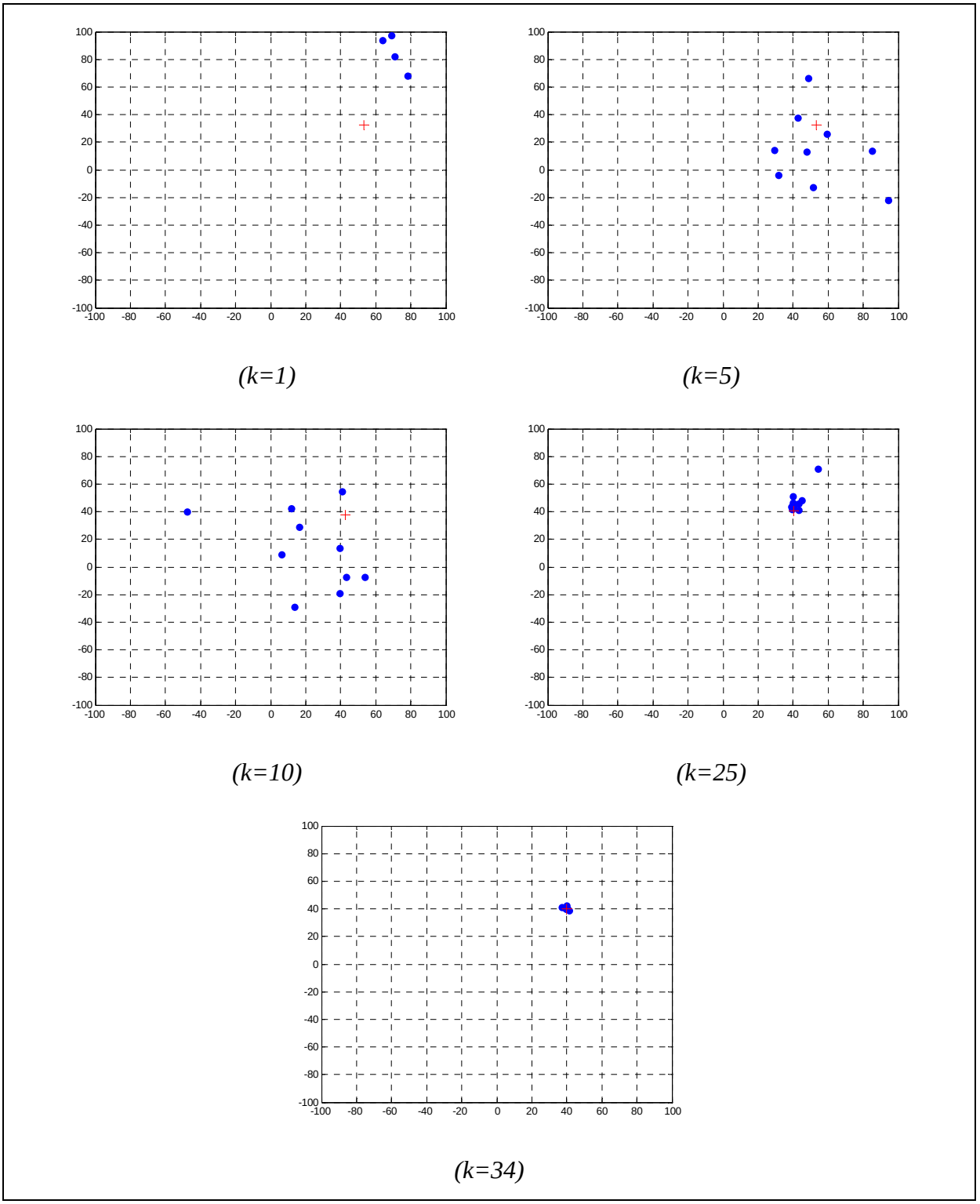


Figura 3.6 – *Representação gráfica da simulação do PSO de população única, para um fi1=fi2=0,5 no MATLAB.*

Note que na Figura3.6 a população da simulação encontrou o seu ponto ótimo em um *k=34*, ou seja houve um total de 34 iterações. É possível observar que as partículas

convergiram próximas ao ponto *x(i)=40, y(i)=40*, ou seja, conseguiram alcançar a solução ótima do problema de minimização, com um erro máximo na distância de 0,01.

Nas simulações realizadas o erro não diminui de maneira contínua, isso ocorre devido ao fato de que ele só diminui quando a população acha um Gbest melhor do que o anterior. Desse modo, é possível concluir que o erro diminui de maneira descontínua, como é ilustrado na Figura 3.7 (observe o salto de *k=6* a *k=7)*. Os dados deste gráfico foram coletados pela função *tic* e *toc* do MATLAB, na qual a cada simulação retorna o tempo de execução de um M-File. Para observar o banco de dados vide o Anexo 7.

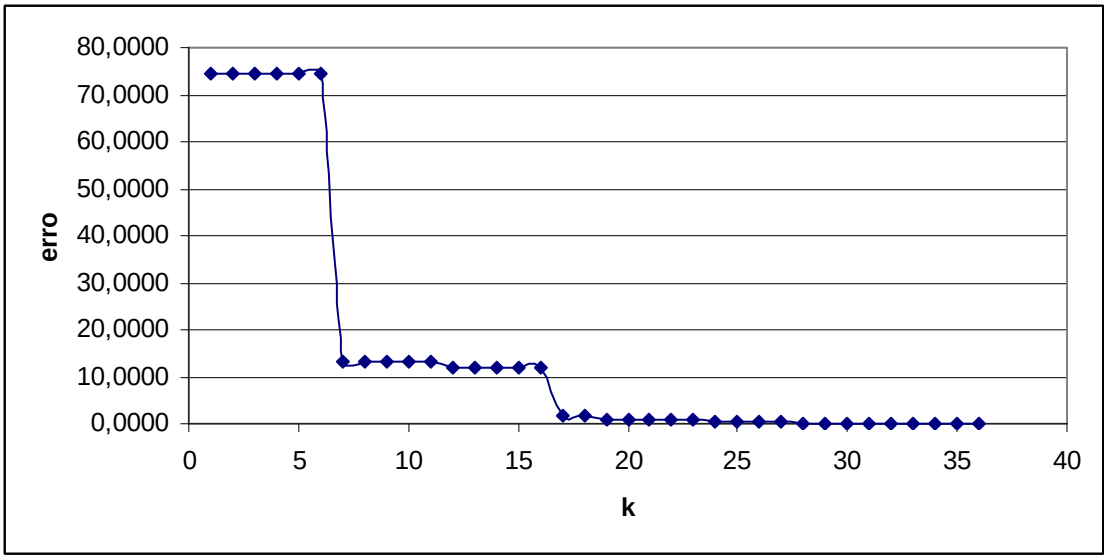


Figura 3.7 – *Gráfico da relação erro e número de itereações.*

Para observar detalhadamente esta programação no MATLAB vide o Anexo 3.

Duas Populações Independentes

Na programação de duas populações independentes em comparação com a população única, foi necessária a duplicação de alguns algoritmos. As linhas indicadas na programação abaixo são os algoritmos que precisavam ser duplicadas de maneira que seja possível executar o programa de duas populações. Isso pode ser verificado em (1), por exemplo, onde podemos ver *fi11=0.5*, *fi21=0.5*, *fi12=0.5* e *fi22=0.5*. As variáveis *fi11=0.5* e *fi21=0.5* correspondem às taxas de cognição da primeira população, enquanto que *fi21=0.5* e *fi22=0.5* são da segunda população.

```
1 -  tic
2
3 -  clear all
4
5 -  fi11=0.5; %-------------------------------------------------------------------(1)
6 -  fi21=0.5;
7 -  fi12=0.5;
8 -  fi22=0.5;
```

Nas linhas (4) e (7) são representadas as funções de cada população, na qual *f(x1(i),y1(i))* corresponde à primeira população e *f(x2(i),y2(i))* à segunda. Observe que em (4) a solução de mínimo é o ponto na qual *x1(i)=0* e *y1(i)=0*, porém em (7) é o ponto cuja as coordenadas são *x2(i)=40* e *y2(i)=40*. Portanto, espera-se que as duas populações tendam a convergir nessas posições.

```
12
13 -  for i=1:10
14
15 -      x1(i)=(100*rand); %-----------------------------------------------------(3)
16 -      y1(i)=(100*rand);
17 -      vx1(i)=(100*rand);
18 -      vy1(i)=(100*rand);
19
20 -      f1(i)=x1(i)^2+y1(i)^2; %-----------------------------------------------(4)
21
22 -      px1(i)=x1(i); %--------------------------------------------------------(5)
23 -      py1(i)=y1(i);
24
25 -      x2(i)=(100*rand); %----------------------------------------------------(6)
26 -      y2(i)=(100*rand);
27 -      vx2(i)=(100*rand);
28 -      vy2(i)=(100*rand);
29
30 -      f2(i)=(x2(i)-40)^2+(y2(i)-40)^2; %-------------------------------------(7)
31
32 -      px2(i)=x2(i);
33 -      py2(i)=y2(i);
34
35 -      if f1(i)<f1best %------------------------------------------------------(8)
36 -          f1best=f1(i);
37 -          gx1=x1(i);
38 -          gy1=y1(i);
39 -      end
40
41 -      if f2(i)<f2best %------------------------------------------------------(9)
42 -          f2best=f2(i);
43 -          gx2=x2(i);
44 -          gy2=y2(i);
45 -      end
46 -  end
47
48 -  k=1;
49
50 -  erro1=100000; %--------------------------------------------------------(10)
51 -  erro2=100000;
```

As condições de parada do programa estão especificadas nas linhas (11) e (12), que estão relacionadas com o *erro* e o número máximo de iterações, respectivamente. A diferença da condição de parada pelo *erro*, em (11), com a programação de população única é o fato de que ocorre a finalização do programa se uma das populações atingir o ponto ótimo, mesmo que haja um *erro* de 0,01. Já em (12) a única diferença foi a duplicação da conseqüência, o *erro* de cada população (*erro1* e *erro2*, na qual correspondem à primeira e segunda população, respectivamente) torna-se 0, quando o número de iterações excede de 100 unidades.

```
52
53 -  while (erro1>0.01)&&(erro2>0.01) %-----------------------------------------(11)
54
55 -      if k>100 %-------------------------------------------------------------(12)
56 -          erro1=0;
57 -          erro2=0;
58 -      else
59 -          for i=1:10 %-----------------------------------------------------(13)
60
61 -              vvx1=vx1(i)+(fi11*rand)*(px1(i)-x1(i))+(fi12*rand)*(gx1-x1(i)); %(14)
62 -              vvy1=vy1(i)+(fi11*rand)*(py1(i)-y1(i))+(fi12*rand)*(gy1-y1(i));
63 -              xx1(i)=x1(i)+vvx1;
64 -              yy1(i)=y1(i)+vvy1;
65
66 -              f1(i)=x1(i)^2+y1(i)^2;
67
68 -              vvx2=vx2(i)+(fi12*rand)*(px2(i)-x2(i))+(fi22*rand)*(gx2-x2(i)); %(15)
69 -              vvy2=vy2(i)+(fi12*rand)*(py2(i)-y2(i))+(fi22*rand)*(gy2-y2(i));
70 -              xx2(i)=x2(i)+vvx2;
71 -              yy2(i)=y2(i)+vvy2;
72
73 -              f2(i)=(x2(i)-40)^2+(y2(i)-40)^2;
74
75 -              px1(i)=x1(i); %---------------------------------------------(16)
76 -              py1(i)=y1(i);
77
78 -              px2(i)=x2(i);
79 -              py2(i)=y2(i);
80
81 -          if f1(i)<f1best %-------------------------------------------------(17)
82 -              f1best=f1(i);
83 -              gx1=x1(i);
84 -              gy1=y1(i);
85 -          end
86
87 -          x1(i)=xx1(i); %---------------------------------------------------(18)
88 -          y1(i)=yy1(i);
89 -          vx1(i)=vvx1;
90 -          vy1(i)=vvy1;
91
92 -          if f2(i)<f2best %-------------------------------------------------(19)
93 -              f2best=f2(i);
94 -              gx2=x2(i);
95 -              gy2=y2(i);
96 -          end
97
98 -          x2(i)=xx2(i); %---------------------------------------------------(20)
99 -          y2(i)=yy2(i);
100 -         vx2(i)=vvx2;
101 -         vy2(i)=vvy2;
102
103 -         end
```

As partículas são inseridas no gráfico representativo pelas linhas (21) e (22), de maneira que é possível diferenciá-las pelas cores, na qual as da primeira população possuem a cor azul, conforme *'.b'*, e as da segunda são vermelhas, segundo *'.r'*. Observe que a

disposição das partículas da primeira população, (21), foi colocada antes do *hold on*, enquanto que a da segunda, (22), foi posta depois do *hold on*. Isso foi feito para que seja possível visualizar ambas as populações, ao mesmo tempo. Caso ambas, (21) e (22), sejam programadas antes desse *hold on*, observaríamos as iterações de somente uma única população.

Note que nas linhas (23) e (24) são plotadas uma cruz em cada uma dessas linhas. Elas representam o Gbest de cada população e também são diferenciadas pelas cores, na qual a cruz da primeira população tem a cor vermelha (‘+*r*’) e a da segunda a cor preta (‘+*k*’).

```
104
105 -      plot(x1(:),y1(:),'.b', 'markersize',20) %--------------------------------(21)
106 -      axis([-100 100 -100 100])
107 -      hold on
108 -      plot(x2(:),y2(:),'.r', 'markersize',20) %--------------------------------(22)
109 -      plot(gx1,gy1,'+r', 'markersize', 10) %-----------------------------------(23)
110 -      plot(gx2,gy2,'+k', 'markersize', 10) %-----------------------------------(24)
111 -      axis([-100 100 -100 100])
112 -      pause(0.1)
113 -      grid
114 -      hold off
115
116 -      erro1=abs(f1best-0); %-------------------------------------------------(25)
117 -      erro2=abs(f2best-0);
118
119 -      k=k+1;
120
121 -      end
122
123 -      pause
124
125 -  end
126
127 -  toc
```

A Figura 3.8 mostra uma simulação da programação descrita acima:

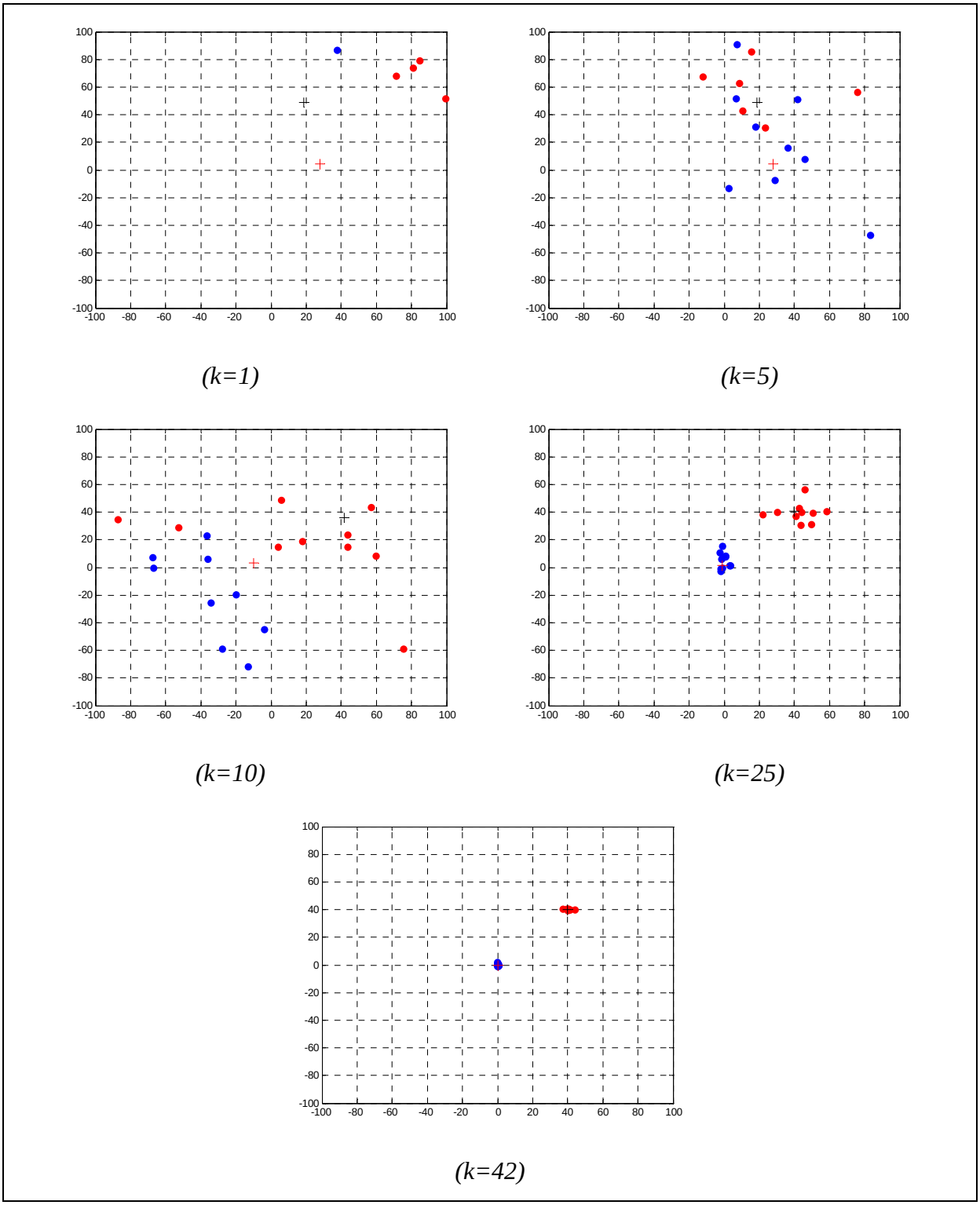


Figura 3.8 – *Representação gráfica da simulação do PSO de duas populações independentes, para um fi11=fi12=fi21=fi22=0,5 no MATLAB.*

Note que na Figura 3.7 houve um total de 42 iterações, ou seja, *k=42*, até que uma das populações tenha atingido a respectiva solução ótima. Em *k=42* as partículas de cada população convergiram na melhor solução possível, com um *erro* máximo de 0,01.

Para observar detalhadamente esta programação no MATLAB vide o Anexo 4

Duas Populações Interagindo Entre Si

Nesta programação há algumas alterações, quando comparado com a simulação de duas populações independentes, de modo que seja possível a interação das populações. A iteração ocorrerá de modo que a população com as “melhores” informações irá “atrapalhar” a outra. A seguir os algoritmos com essas principais diferenças:

```
tic

clear all

fi1=0.5;
fi2=0.5;
fiy1=0.5;
fiy2=0.5;

f1best=100000;
f2best=100000;

for i=1:10

    x1(i)=(100*rand);
    y1(i)=(100*rand);
    vx1(i)=(100*rand);
    vy1(i)=(100*rand);

    px1(i)=x1(i);
    py1(i)=y1(i);

    x2(i)=(100*rand);
    y2(i)=(100*rand);
    vx2(i)=(100*rand);
    vy2(i)=(100*rand);

    px2(i)=x2(i);
    py2(i)=y2(i);

    f1(i)=x1(i)^2+y1(i)^2; %-------------------------------------------------(1)
    f2(i)=x2(i)^2+y2(i)^2;
```

Em (1) são definidas as funções, *f(x(i),y(i)*, de cada população, contudo, note que a solução ótima, que é de mínimo, é igual para ambas, *x(i)=y(i)=0*.

```
33
34 -       if f1(i)<f1best
35 -           f1best=f1(i);
36 -           gx1=x1(i);
37 -           gy1=y1(i);
38 -       end
39
40 -       if f2(i)<f2best
41 -           f2best=f2(i);
42 -           gx2=x2(i);
43 -           gy2=y2(i);
44 -       end
45
46 -   end
47
48 -   if f1best<f2best %-------------------------------------------------------------(2)
49 -       fi21=0.3;
50 -       fi22=0.3;
51 -   else
52 -       fi11=0.3;
53 -       fi12=0.3;
54 -   end
```

Foi inserida uma função *if*, em (2), com a condição de que se a primeira população tenha encontrado um Gbest "melhor" do que a segunda, *fi21=0,3* e *fi22=0,3*, caso contrário, *fi11=0,3* e *fi12=0,3*. Essas conseqüências fazem com que a população que não tenha a melhor informação sobre a solução ótima na iteração *k* tenha taxas de cognição mais baixas em *k*+1.

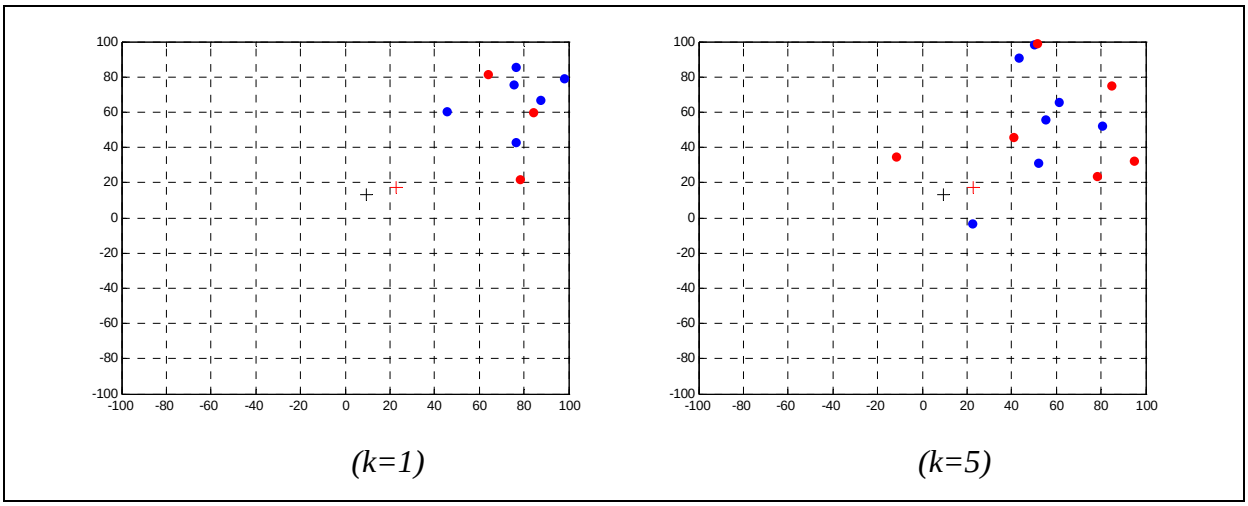
```
55
56 -   k=1;
57
58 -   erro1=100000;
59 -   erro2=100000;
60
61 -   while (erro1>0.1)&&(erro2>0.1)
62
63 -       if k>100
64 -           erro1=0;
65 -           erro2=0;
66 -       else
```

```
67 -            for i=1:10
68
69 -                vvx1=vx1(i)+(fi11*rand)*(px1(i)-x1(i))+(fi12*rand)*(gx1-x1(i));
70 -                vvy1=vy1(i)+(fi11*rand)*(py1(i)-y1(i))+(fi12*rand)*(gy1-y1(i));
71 -                xx1(i)=x1(i)+vvx1;
72 -                yy1(i)=y1(i)+vvy1;
73
74 -                f1(i)=x1(i)^2+y1(i)^2;
75
76 -                vvx2=vx2(i)+(fi21*rand)*(px2(i)-x2(i))+(fi22*rand)*(gx2-x2(i));
77 -                vvy2=vy2(i)+(fi21*rand)*(py2(i)-y2(i))+(fi22*rand)*(gy2-y2(i));
78 -                xx2(i)=x2(i)+vvx2;
79 -                yy2(i)=y2(i)+vvy2;
80
81 -                f2(i)=x2(i)^2+y2(i)^2;
82
83 -                px1(i)=x1(i);
84 -                py1(i)=y1(i);
85
86 -                px2(i)=x2(i);
87 -                py2(i)=y2(i);
88
89 -            if f1(i)<f1best
90 -                f1best=f1(i);
91 -                gx1=x1(i);
92 -                gy1=y1(i);
93 -            end
94
95 -            x1(i)=xx1(i);
96 -            y1(i)=yy1(i);
97 -            vx1(i)=vvx1;
98 -            vy1(i)=vvy1;
99
100 -           if f2(i)<f2best
101 -               f2best=f2(i);
102 -               gx2=x2(i);
103 -               gy2=y2(i);
104 -           end
105
106 -           x2(i)=xx2(i);
107 -           y2(i)=yy2(i);
108 -           vx2(i)=vvx2;
109 -           vy2(i)=vvy2;
110
111 -           end
112
113 -       if f1best<f2best %-------------------------------------------------------(3)
114 -           fi21=0.3;
115 -           fi22=0.3;
116 -       else
117 -           fi11=0.3;
118 -           fi22=0.3;
119 -       end
```

Note que (3) é idêntico a (2), esta repetição ocorre pela razão de se desejar que as duas populações continuem a interagirem entre si, ao longo das iterações.

```
120
121 -       plot(x1(:),y1(:),'.b','markersize',20)
122 -       axis([-100 100 -100 100])
123 -       hold on
124 -       plot(x2(:),y2(:),'.r','markersize',20)
125 -       plot(gx1,gy1,'+r','markersize',10)
126 -       plot(gx2,gy2,'+k','markersize',10)
127 -       axis([-100 100 -100 100])
128 -       pause(0.1)
129 -       grid
130 -       hold off
131
132 -       erro1=abs(f1best-0);
133 -       erro2=abs(f2best-0);
134
135 -       k=k+1;
136
137 -       end
138
139 -       pause
140
141 -   end
142
143 -   toc
144
```

A Figura 3.9 irá mostrar graficamente a simulação da programação descrita acima:



(k=1) *(k=5)*

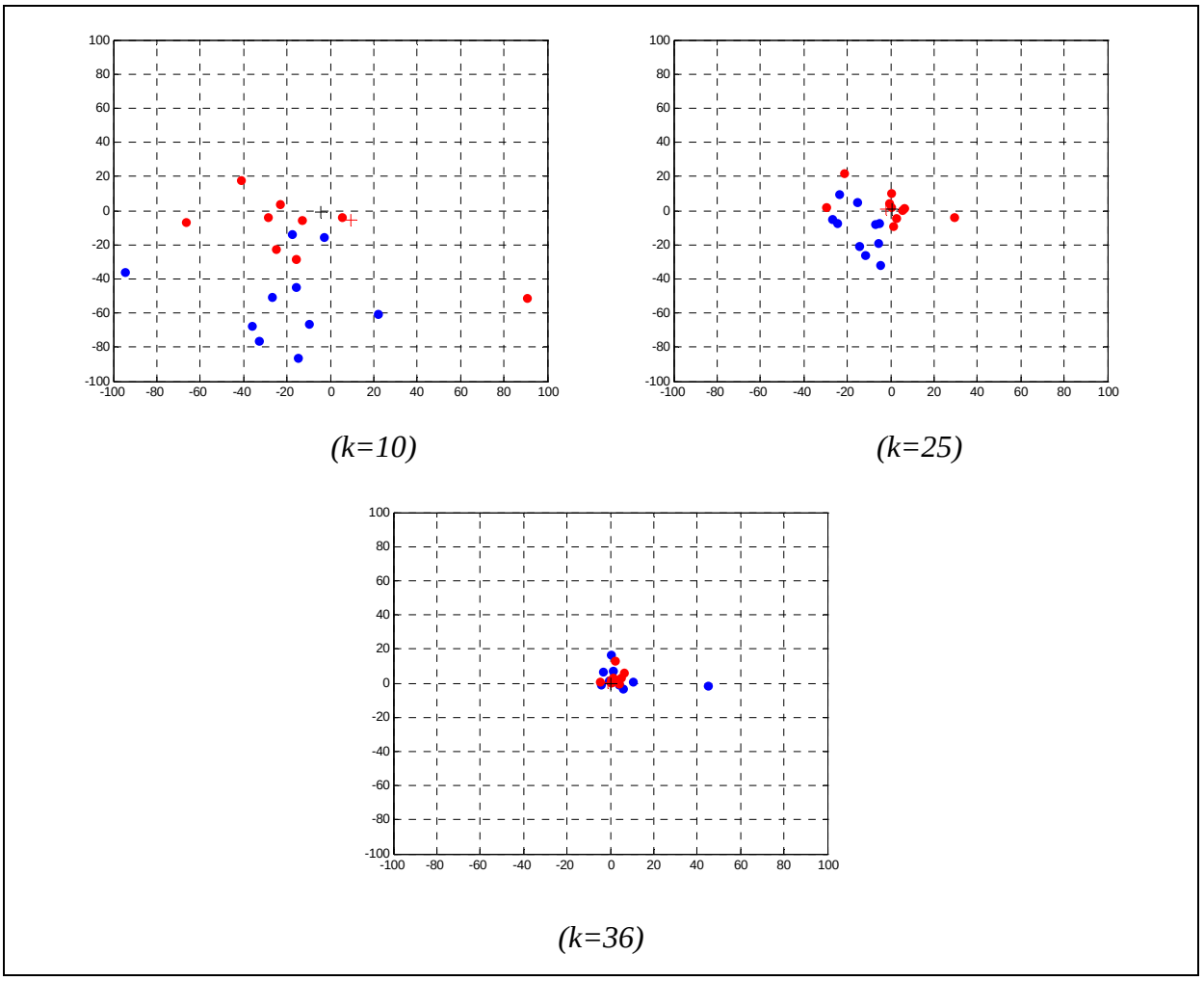


Figura 3.9 – *Representação gráfica da simulação do PSO de duas populações interagindo entre si, para um fi11=fi12=fi21=fi22=0,5 no MATLAB.*

Na Figura 3.9 as partículas da primeira população estão menos concentradas, comparado com as da segunda, na solução ótima. O que nos permite afirmar que, muito provavelmente foi a segunda população que atingiu a melhor solução possível, com um erro máximo de 0,01.

Para observar detalhadamente esta programação no MATLAB vide o Anexo 5

3.3 – Cenários e Simulações

Neste tópico mostramos o que ocorre quando alteramos o *fi1* e o *fi2* das programações descritas em **3.1** e **3.2**.

VBA

Alterandos o *fi1* e *fi2* no programa de uma única população do VBA:

fi1=0.9 e *fi2=0.9*:

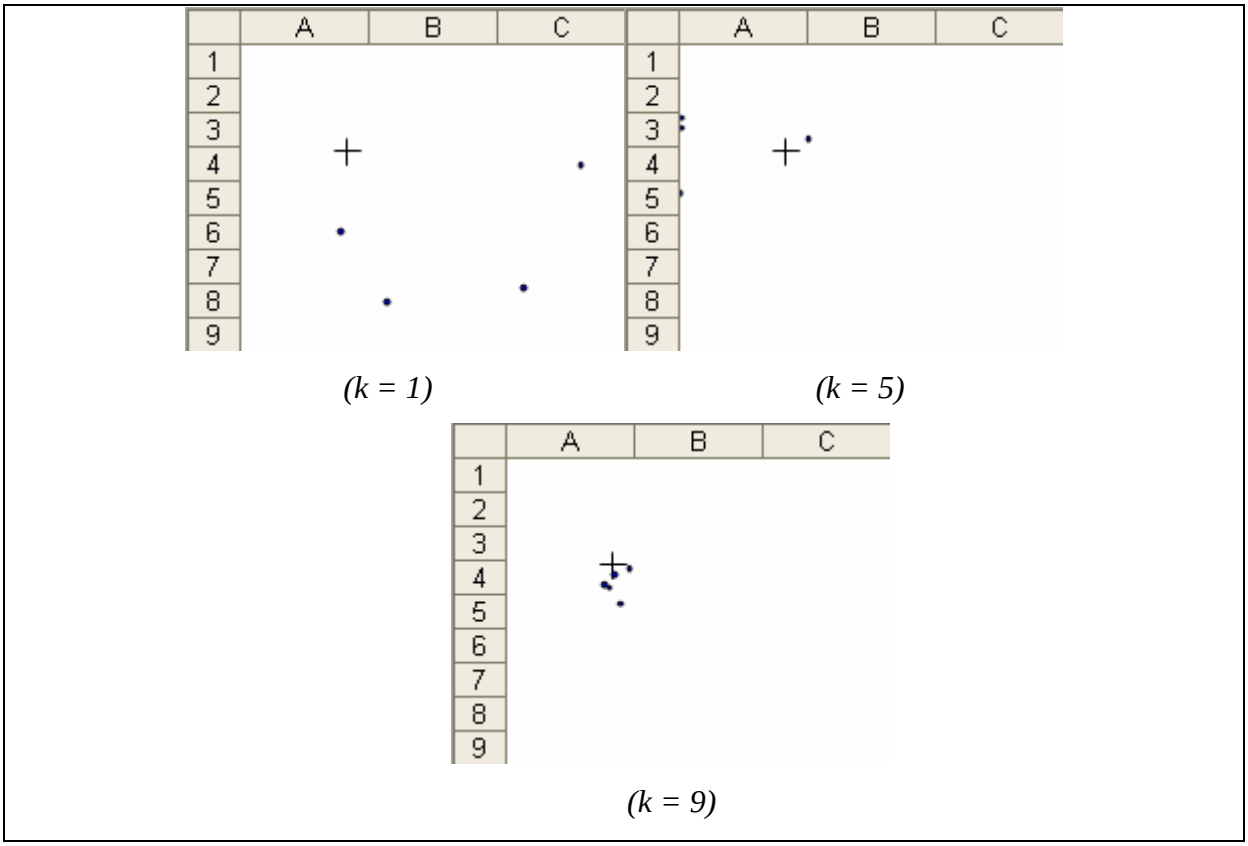


Figura 3.10 – *Representação gráfica da simulação do PSO de população única, para um fi1=fi2=0,9 no VBA.*

Observe que houve uma diferença clara entre o número de iterações necessárias para se alcançar a solução ótima de uma mesma equação da Figura 3.10 para a Figura 3.2. As duas figuras são representações gráficas da simulação de uma mesma programação, contudo com *fi1* e *fi2* diferentes. É possível analisar que uma taxa de cognição maior permite uma convergência mais rápida das partículas em torno do ponto ótimo.

fi1=0.1 e *fi2=0.1*:

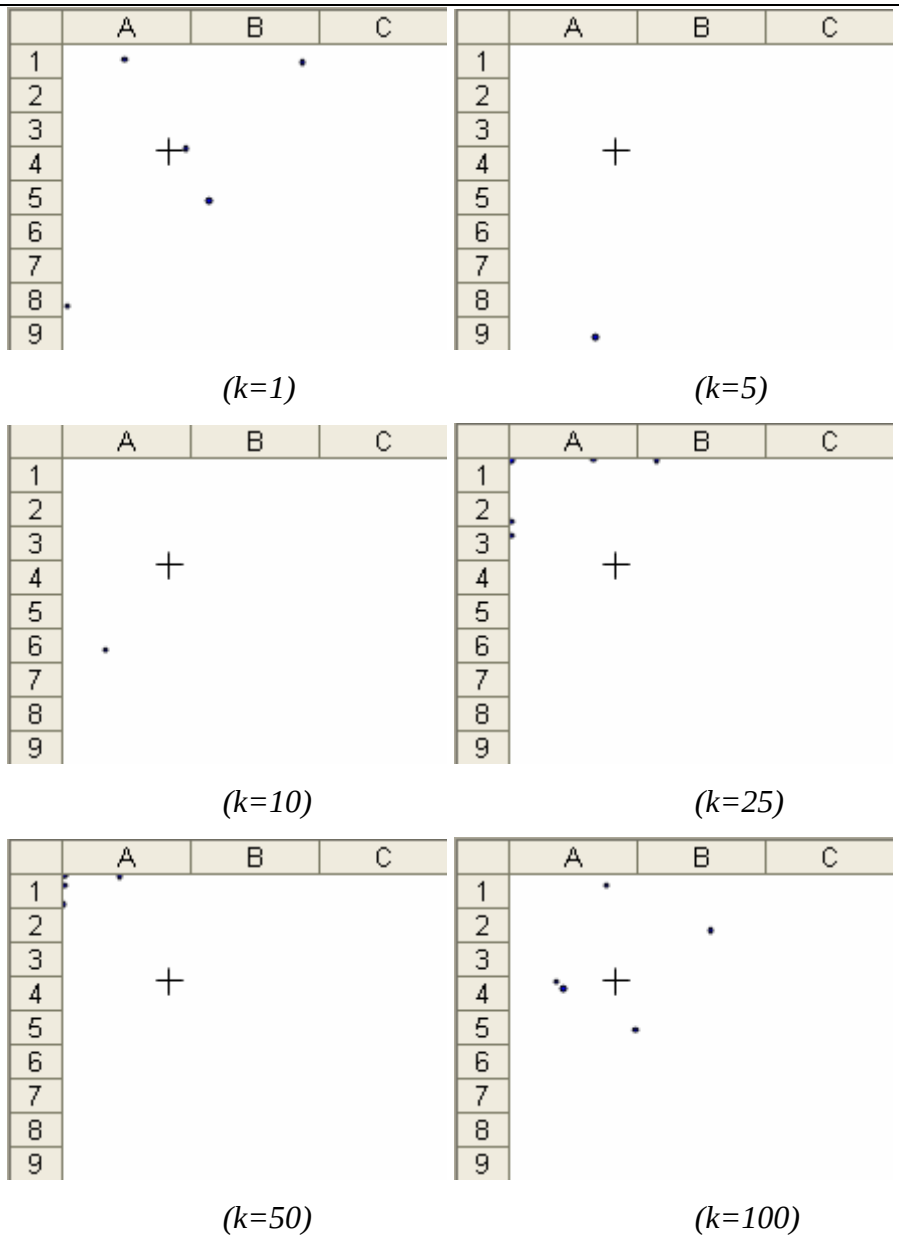


Figura 3.11 – *Representação gráfica da simulação do PSO de população única, para um fi1=fi2=0,1 no VBA.*

Apesar das diferenças entre a Figura 3.11 e a Figura 3.2 serem tão notáveis quanto a da relação anterior (Figura 3.10), ocorre o contrário nesta. Mesmo após 100 iterações não foi possível observar que as partículas tenham convergido no ponto ótimo. Isso mostra que taxas de cognição mais baixas dificultam essa convergência, demandando um maior número de iterações.

MATLAB

fi1=0.9 e *fi2=0.9*

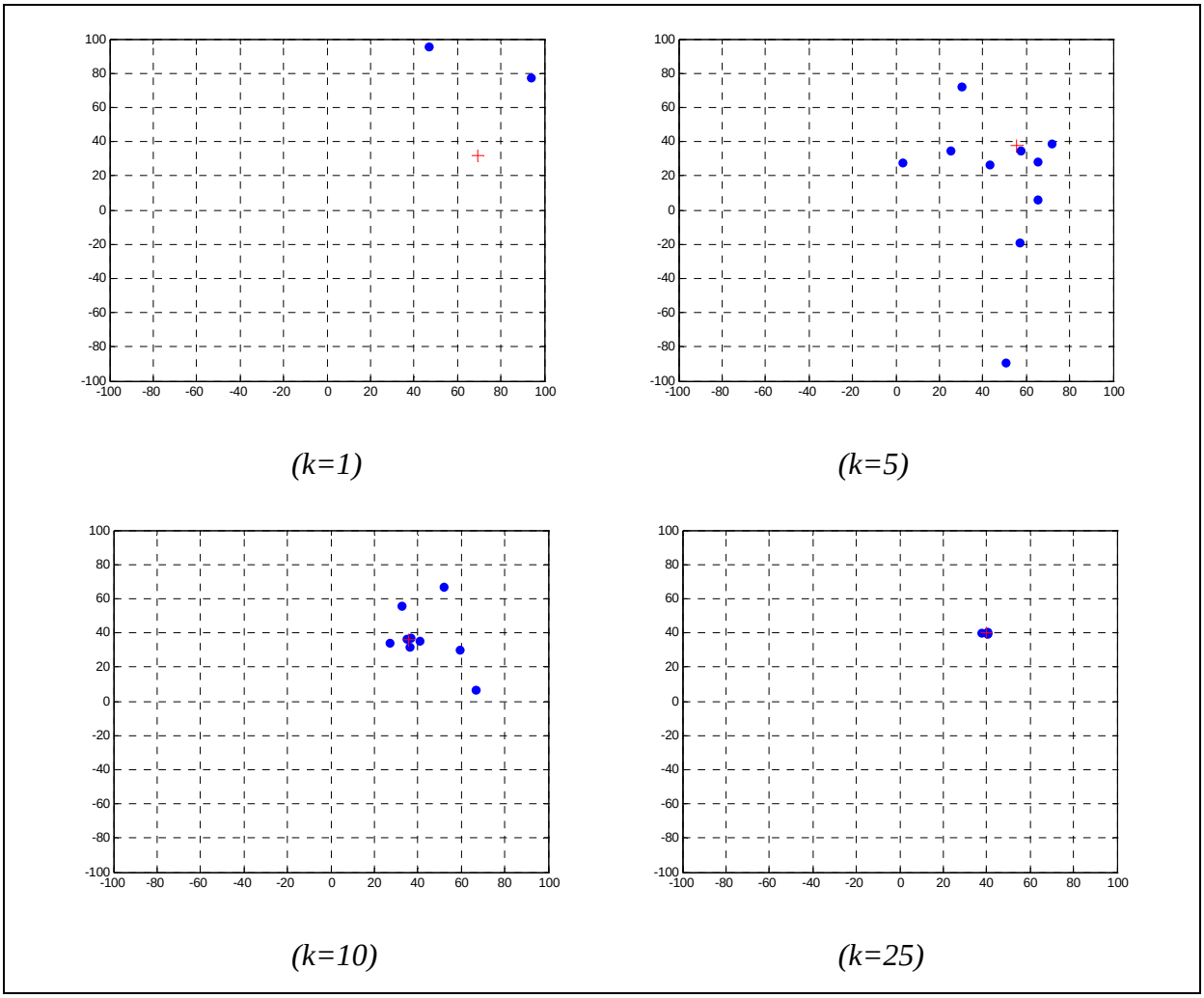


Figura 3.12 – *Representação gráfica da simulação do PSO de população única, para um fi1=fi2=0,9 no MATLAB*

Observe que houve uma diferença clara entre o número de iterações necessárias para se alcançar a solução ótima de uma mesma equação, quando comparamos a Figura 3.12 com a Figura 3.6. As duas figuras são representações gráficas da simulação de uma mesma programação, contudo com *fi1* e *fi2* diferentes. É possível analisar que uma taxa de cognição maior permite uma convergência mais rápida das partículas em torno do ponto ótimo.

fi1=0.1 e *fi2=0.1*

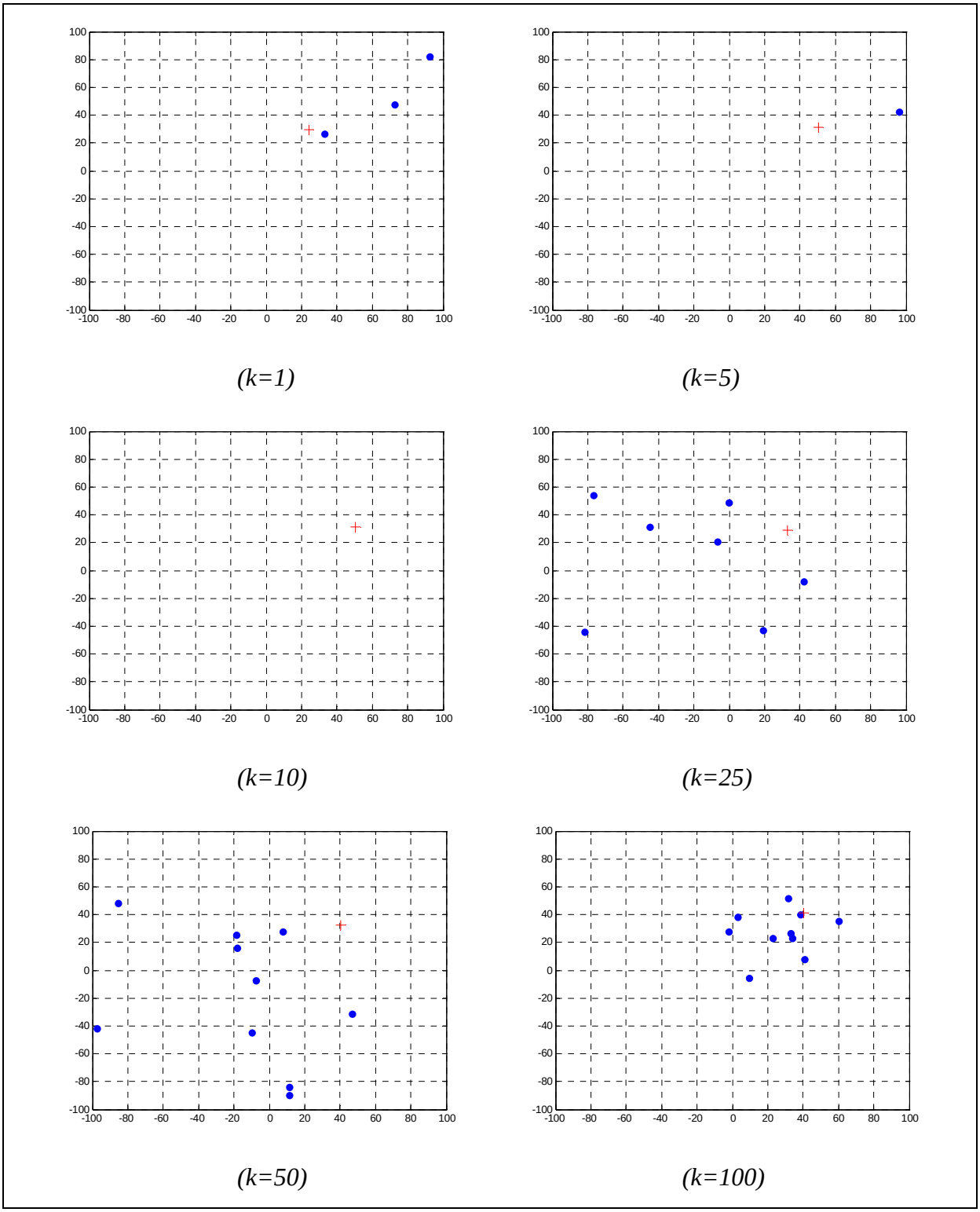


Figura 3.13 – *Representação gráfica da simulação do PSO de população única, para um fi1=fi2=0,1 no MATLAB*

Apesar das diferenças entre a Figura 3.13 e a Figura 3.6 serem tão notáveis quanto a da simulação anterior (Figura 3.12), ocorreu o contrário desta vez, mesmo após 100 iterações não foi possível observar que as partículas tenham convergido no ponto ótimo. Isso mostra

que taxas de cognição mais baixas dificultam essa convergência, demandando um maior número de iterações.

A Figura 3.14 mostra diferentes simulações com três *fi1* e *fi2*, cada uma. Vamos definir nesta sessão que as simulações com *fi1=fi2=0,5* são do tipo 1, as com *fi1=fi2=0,*9 são do tipo 2 e as com *fi1=fi2=0,*1 como as do tipo 3. Note que as simulações do tipo 1 mantêm um tempo de execução em torno de 4 a 6, as do tipo 2 em torno de 2 a 4, e, por fim, as do tipo 3 ficaram em torno de 12 a 14. Observe ainda que a variância da simulação do tipo 3 foi menor do que a dos demais tipos.

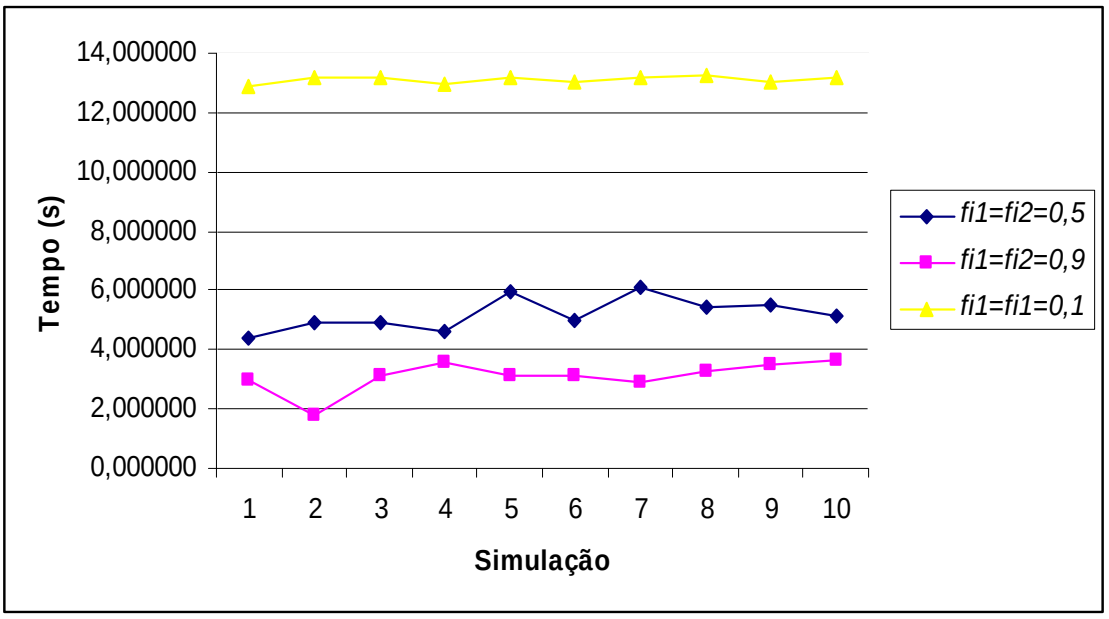


Figura 3.14 – *Gráfico que representa o tempo que cada tipo de simulação, ou seja, para fi diferentes, leva até alcançar o ponto ótimo.*

3.4 – Restrições das Trajetórias

Neste tópico analisaremos descritivamente uma programação feita com a linguagem do MATLAB, muito próximo ao que foi visto no tópico **3.2**. No entanto, a principal diferença é que impomos restrições quanto à posição das partículas na simulação. O movimento delas é delimitado por duas retas paralelas, de maneira que só será possível se deslocarem entre as retas. Este programa foi baseado nas seguintes partes principais dos algoritmos abaixo:

```
1 -  tic
2
3 -  clear all
4
5 -  b1=0; %-------------------------------------------------(1)
6 -  b2=-100;
7
8 -  xl10=200; %-------------------------------------------------(2)
9 -  yl10=400;
10 -  xl11=-200;
11 -  yl11=-400;
12
13 -  a=(yl11-yl10)/(xl11-xl10); %-----------------------------(3)
14
15 -  xl20=xl10; %-------------------------------------------------(4)
16 -  yl20=xl20*a+b2;
17 -  xl21=xl11;
18 -  yl21=xl21*a+b2;
19
20 -  fi1=0.5;
21 -  fi2=0.5;
22
23 -  fbest=100000000;
```

Em (1), *b1* e *b2* correspondem ao coeficiente linear de duas retas, respectivamente. Foram definidas como 0 e -100 por conveniência na hora de visualizar o movimento das partículas.

Na parte (2), *xl10* e *yl10* é o ponto inicial do segmento de reta 1, enquanto que *yl11* e *yl11* é o ponto final do segmento de reta 1. A explicação para os valores em cada um dos eixos de seus respectivos pontos será feita mais adiante.

A linha (3) mostra os cálculos para definir *a*, que é o coeficiente angular das retas. Note que ela depende dos valores definidos em (2). Foi calculado um único coeficiente angular para que as duas retas a serem formadas sejam paralelas. (Vide o **Apêndice A**)

Uma vez que foi determinado o *a*, foi possível desenvolver, em (4), a reta que passa nos pontos (*xl20*, *yl20)* e (*xl21*, *yl21*), na qual *xl20* e *yl20* é o ponto inicial do segmento de reta 2 e *xl21* e *yl21* é o ponto final do segmento de reta 2. Note que para este cálculo foi usado o *a* definido pela reta 1, ou seja, em (3). (Vide o **Apêndice A**)

```
24
25 -  for i=1:10
26 -      x(i)=(100*rand); %-----------------------------------(5)
27 -      y(i)=(100*rand);
28
29 -      while (x(i)<(y(i)-b1)/a)||(x(i)>(y(i)-b2)/a) %-------(6)
30 -      x(i)=(100*rand); %-----------------------------------(7)
31 -      end
32 -      while (y(i)>(x(i)*a+b1))||(y(i)<(x(i)*a+b2)) %-------(8)
33 -      y(i)=(100*rand); %-----------------------------------(9)
34 -      end
35
36 -      vx(i)=(100*rand); %----------------------------------(10)
37 -      vy(i)=(100*rand);
38
39 -      f(i)=(x(i)+125)^2+(y(i)+300)^2; %--------------------(11)
40
41 -      px(i)=x(i);
42 -      py(i)=y(i);
43
44 -      if f(i)<fbest
45 -          fbest=f(i);
46 -          gx=x(i);
47 -          gy=y(i);
48 -      end
49 -  end
```

Para que a posição inicial das partículas não seja muito próxima da equação definida em (11), *x(i)* e *y(i)* foram multiplicados por 100, como em (5).

Para delimitar a posição inicial das partículas, de modo que fiquem entre as retas, foi utilizada uma função *While*, como em (6) e (8). Esta função foi programada em duas etapas:

1. Se a partícula gerada estiver em uma posição no eixo x à esquerda do segmento de reta 1 ou à direita do segmento de reta 2 ela será gerada novamente, como em (7). Esse processo será repetido infinitas vezes até que a condição de parada seja alcançada.
2. Após o processo descrito acima, a mesma partícula será gerada infinitas vezes até que a condição de parada em (9) seja satisfeita. Essa condição é a de que a posição inicial da partícula no eixo y deverá estar abaixo do segmento de reta 1 e acima do segmento de reta 2.

Em (10), são geradas as velocidades iniciais de cada partícula nos seus respectivos eixos.

A linha (11) é a equação a ser otimizada, uma minimização neste caso, desta simulação. Note que o ponto ótimo será em *f*(-125,-300), ou seja, em um ponto, cujos valores em cada eixo fazem com que função seja igual a zero.

```
50
51 -  k=1;
52 -  erro=100000;
53
54 -  while erro>0.01
55
56 -      if k>100
57 -          erro=0;
58 -      else
59 -          for i=1:10
60 -              vvx=vx(i)+(fi1*rand)*(px(i)-x(i))+(fi2*rand)*(gx-x(i));
61 -              vvy=vy(i)+(fi1*rand)*(py(i)-y(i))+(fi2*rand)*(gy-y(i));
62 -              xx=x(i)+vvx;
63 -              yy=y(i)+vvy;
64
65 -              f(i)=(x(i)+125)^2+(y(i)+300)^2;
66
67 -              px(i)=x(i);
68 -              py(i)=y(i);
69
70 -          if f(i)<fbest
71 -              fbest=f(i);
72 -              gx=x(i);
73 -              gy=y(i);
74 -          end
75
```

```
76 -            if (xx>(yy-b1)/a) %------------------------------(12)
77 -                x(i)=xx;
78 -            else
79 -                xx=(yy-b1)/a;
80 -                x(i)=xx;
81 -            end
82
83 -            if (xx<(yy-b2)/a) %------------------------------(13)
84 -                x(i)=xx;
85 -            else
86 -                xx=(yy-b2)/a;
87 -                x(i)=xx;
88 -            end
89
90 -            if (yy<(xx*a+b1)) %------------------------------(14)
91 -                y(i)=yy;
92 -            else
93 -                yy=xx*a+b1;
94 -                y(i)=yy;
95 -            end
96
97 -            if (yy>(xx*a+b2)) %------------------------------(15)
98 -                y(i)=yy;
99 -            else
100 -               yy=xx*a+b2;
101 -               y(i)=yy;
102 -           end
103
104 -           vx(i)=vvx;
105 -           vy(i)=vvy;
106 -       end
```

As partes (12), (13), (14) e (15) são funções *if* que impedem que partículas violem as restrições ao longo das iterações. Por exemplo, em (12), se a posição gerada pela equação de PSO da partícula *(i)* no eixo x estiver à direita do segmento de reta 1, será a nova posição da partícula na iteração seguinte, caso contrário, a nova posição será igual ao ponto acima do segmento de reta 1 mais próximo entre este mesmo segmento de reta e aquela gerada pela equação de PSO. (12) e (13), são as linhas que não permitirão a violação da restrição imposta pelo segmento de reta 1, enquanto que (14) e (15) são as linhas que impedem a ultrapassagem para abaixo do segmento de reta 2 pelas partículas.

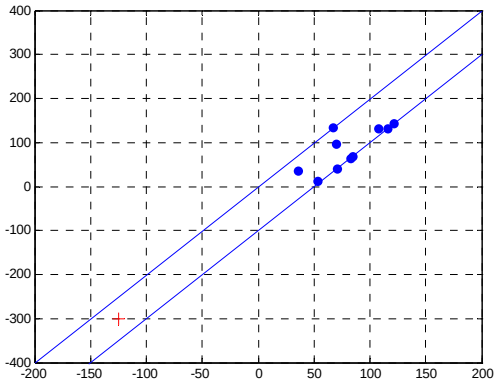
```
107
108 -       plot(x,y,'.', 'markersize', 20)
109 -       axis([xl11 xl10 yl11 yl10]) %------------------------(16)
110 -       hold on
111 -       line([xl11 xl10],[yl11 yl10],[1 1],'LineStyle','-') %(17)
112 -       line([xl21 xl20],[yl21 yl20],[1 1],'LineStyle','-') %(18)
113 -       plot(-125,-300,'+r', 'markersize', 10) %-------------(19)
114 -       axis([xl11 xl10 yl11 yl10])
115 -       pause(0.1)
116 -       grid
117 -       hold off
118
119 -       erro=abs(fbest-0);
120 -       k=k+1;
121
122 -       end
123
124 -       pause
125
126 -   end
127
128 -   toc
```

A explicação para os valores em (2) é feita em (16), onde é a linha que define a dimensão do gráfico representativo, ou seja, os valores em (2) foram escolhidos de modo que facilitem a visualização desta simulação.

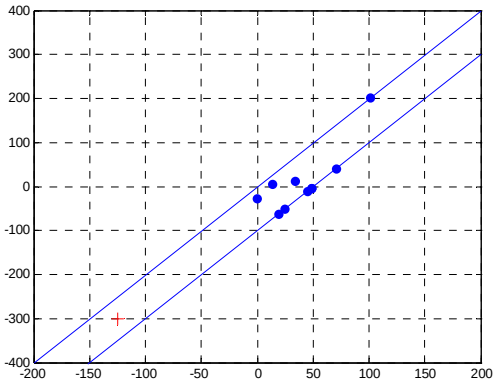
As linhas (17) e (18) são as funções que geram os segmentos de retas 1 e 2, respectivamente. Eles são preenchidos pelos valores descritos em (2) e (4), de maneira que os valores no primeiro colchete representam a posição inicial e final, nesta ordem, no eixo x, o segundo é a posição inicial e final, nesta ordem, no eixo y. *'LineStyle'* define o tipo de reta a ser gerado no gráfico, que é '-', ou seja, contínua. Note que dessa maneira os coeficientes lineares definidos em (2) determinam a amplitude entre os dois segmentos de retas.

A linha (19) gera uma cruz de cor vermelha no ponto (-125, -300), ou seja, no ponto ótimo da equação definida em (11).

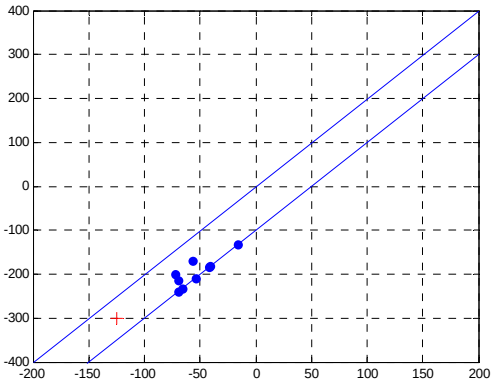
A Figura 3.13 mostra graficamente a simulação da programação descrita nesta seção.



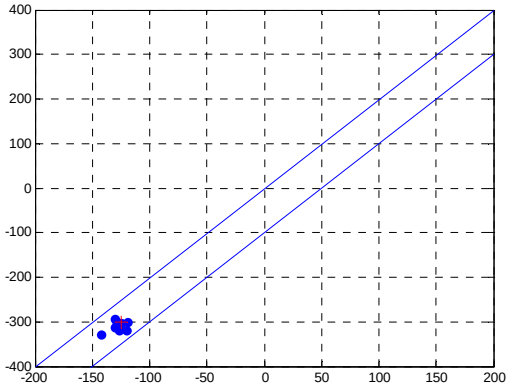
(k=1)



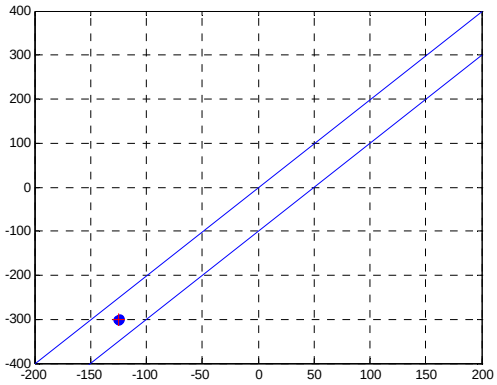
(k=5)



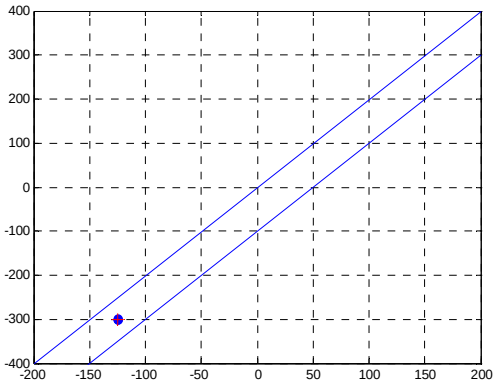
(k=10)



(k=25)



(k=50)



(k=52)

Figura 3.13 – *Representação gráfica da simulação do PSO de população única com restrições, para um fi1=fi2=0,5.*

Observando a Figura 3.13 notamos que em nenhum momento houve violação das restrições por parte das partículas, e mesmo assim elas convergiram no ponto de solução ótimo. Além disso, o número de iterações foi maior do que 50 (*k>50)* pela primeira vez

quando usamos *fi1=fi2=0,5*, indicando que as restrições podem fazer com que as partículas demorem mais para convergirem no ponto ótimo.

Para observar detalhadamente esta programação no MATLAB vide o Anexo 6

Capítulo 4 – Simulações no Mercado Financeiro

Para verificarmos se as equações de PSO são aplicáveis no mercado financeiro foi feito, inicialmente, um gráfico, vide Figura 4.1, do volume negociado de vários dias da ação PETR4 em relação ao VALE3, ambas ponderadas pelo volume da IBOV. Estes dados foram retirados do programa ECONOMÁTICA, salvos em um arquivo do Excel e importados para o MATLAB de maneira que seja possível a visualização do gráfico abaixo.

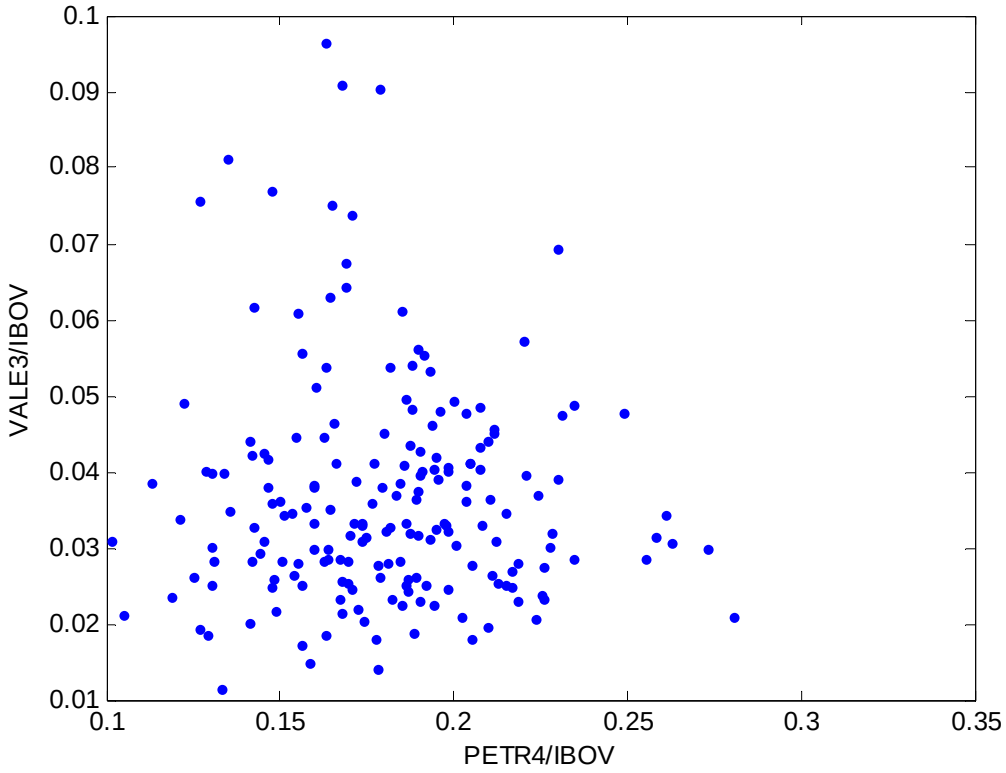


Figura 4.1 – *Relação do volume negociado da PETR4 e a VALE3.*

Na Figura 4.1 é possível observar um ponto de convergência dos volumes negociados, mesmo que os dados estejam bem dispersos, em *(0.154 ,0.034)*.

Para observar detalhadamente o banco de dados utilizado para o gráfico da Figura 4.1 vide o Anexo 9.

Conclusões Preliminares

No capítulo **3** demonstrou-se a aplicação e a simulação do PSO no VBA e no MATLAB. Desta maneira, foi observado que o MATLAB possui ferramentas que facilitam a programação deste tipo de simulação, além de que é um programa muito mais rápido (a simulação no VBA parecia uma seqüência de pausa (*pause*) e avanço (*play*)). Além disso, no MATLAB é possível utilizar as funções *tic* e *toc*, que mostram o tempo de execução de um M-file, e a função *grid*, que facilita mais a visualização das posições da partículas e do ponto de solução ótima nos gráficos representativos.

Após as simulações feitas em **3.4** é possível afirmar que uma simulação de PSO pode ser feita mesmo com restrições quanto à posição das partículas. As partículas continuam a procurar pela melhor solução possível, ou seja, a solução ótima dentro das restrições estabelecidas.

Segundo as informações do **Capítulo 4**, podemos afirmar que é possível observar pontos de convergência no volume negociado em um dia no mercado de ativos na BOVESPA, contudo, ainda é muito cedo para afirmar a aplicabilidade do PSO nesse ambiente.

Referências bibliográficas

LEE, Kwang Y.; EL-SHARKAWI, Mohamed A.. **Modern Heuristic Optimization Techniques:** Theory and Applications to Power Systems. New Jersey: Wiley, 2008. 586 p.

BONABEAU, Eric; DORIGO, Marco; THERAULAZ, Guy. **Swarm Intelligence:** From Natural to Artificial Systems. New Mexico: Oxford University Press, 1999. 307 p.

CHANG, Hyeong Soo. **Computational Intelligence Optimization:** Particle Swarm Optimization and Ant Colony Optimization, A Gentle Introduction. *Invited Talk for the Graduate Seminar*, Department of Electronic and Electrical Engineering, POSTECH, 9., 2005.

KENDALL, Graham; SU, Yan. Artificial Intelligence and Applications. In: IASTED, 23., 2005, Innsbruck. **A Particle Swarm Optimization Approach in the Construction of Optimal Risky Portfolios** . Innsbruck: , 2005, 6 p.

Apêndice A

Equação da reta:

$$y = ax + b$$

Coeficiente angular de uma reta:

$$a = \frac{\Delta y}{\Delta x}$$

Anexo 1 – Programação no VBA do PSO de Uma População

```
Private Sub CommandButton1_Click()

Dim W As Worksheet
Dim i As Integer
Dim x(5) As Single
Dim y(5) As Single
Dim xbest(5) As Single
Dim ybest(5) As Single
Dim pto As Shape
Dim vx(5) As Single
Dim vy(5) As Single
Dim vvx As Single
Dim vvy As Single
Dim gx As Single
Dim gy As Single
Dim xx As Single
Dim yy As Single
Dim f(5) As Single
Dim fbest As Single
Dim fi1 As Single
Dim fi2 As Single

Range(Cells(1, 1), Cells(100, 100)).delete '----------------------------------------------(1)

Range(Cells(1, 1), Cells(100, 100)).Interior.Color = vbWhite

fi1 = 0.5 '-------------------------------------------------------------------------------(2)
fi2 = 0.5
```

```
fbest = 100000 '-------------------------------------------------------------------------(3)

Set W = Worksheets("Plan1") '-----------------------------------------------------------(4)

W.Shapes.AddLine(40, 35, 40, 45).Select '-----------------------------------------------(5)
W.Shapes.AddLine(35, 40, 45, 40).Select

Randomize  '-----------------------------------------------------------------------------(6)

For i = 1 To 5

   x(i) = (100 * Rnd) '-------------------------------------------------------------(7)
   y(i) = (100 * Rnd)

   vx(i) = (100 * Rnd) '------------------------------------------------------------(8)
   vy(i) = (100 * Rnd)

   f(i) = (x(i) - 40) ^ 2 + (y(i) - 40) ^ 2 '---------------------------------------(9)

   With W.Shapes.AddShape(msoShapeOval, x(i), y(i), 2, 2) '-----------------------(10)
   .Fill.ForeColor.RGB = RGB(0, 0, 255)
   End With

      xbest(i) = x(i) '-------------------------------------------------------(11)
      ybest(i) = y(i)

   If f(i) < fbest Then '--------------------------------------------------------(12)
      fbest = f(i)
      gx = x(i)
      gy = y(i)
   End If

Next i
```

```
k = 1 '----------------------------------------------------------------------------(13)

erro = 100000 '-------------------------------------------------------------------(14)

Do While erro > 0.01 '-------------------------------------------------------------(15)

   If k > 100 Then '-----------------------------------------------------------(16)
      erro = 0
   Else

      For i = 1 To 5 '-------------------------------------------------------(17)

         vvx = vx(i) + (fi1 * Rnd) * (xbest(i) - x(i)) + (fi2 * Rnd) * (gx - x(i)) '---(18)
         vvy = vy(i) + (fi1 * Rnd) * (ybest(i) - y(i)) + (fi2 * Rnd) * (gy - y(i))

         xx = vvx + x(i) '-----------------------------------------------------(19)
         yy = vvy + y(i)

         f(i) = (x(i) - 40) ^ 2 + (y(i) - 40) ^ 2 '----------------------------------(20)

         xbest(i) = x(i) '-----------------------------------------------------(21)
         ybest(i) = y(i)


         If f(i) < fbest Then '-----------------------------------------------(22)
            fbest = f(i)
            gx = x(i)
            gy = y(i)
         End If

         x(i) = xx '-----------------------------------------------------------(23)
         y(i) = yy
         vx(i) = vvx
         vy(i) = vvy
```

```
    Next i

  End If

  For Each pto In ActiveSheet.Shapes '-----------------------------------------------(24)
    pto.delete
  Next pto

  W.Shapes.AddLine(40, 35, 40, 45).Select
  W.Shapes.AddLine(35, 40, 45, 40).Select

  For i = 1 To 5  '-------------------------------------------------------------------(25)
    With W.Shapes.AddShape(msoShapeOval, x(i), y(i), 2, 2)
    .Fill.ForeColor.RGB = RGB(0, 0, 255)
    End With
  Next i

  Application.Wait (Now + TimeValue("0:00:01")) '-------------------------------------(26)

  k = k + 1 '-------------------------------------------------------------------------(27)
  erro = Abs(fbest - 0)

Loop '-----------------------------------------------------------------------------(28)
End

End Sub
```

Anexo 2 – Programação no VBA do PSO de Duas Populações Independentes

```
Private Sub CommandButton1_Click()

Dim W As Worksheet
Dim i As Integer
Dim x(5) As Single
Dim y(5) As Single
Dim x2(5) As Single
Dim y2(5) As Single
Dim xbest(5) As Single
Dim ybest(5) As Single
Dim x2best(5) As Single
Dim y2best(5) As Single
Dim pto As Shape
Dim vx(5) As Single
Dim vy(5) As Single
Dim vvx As Single
Dim vvy As Single
Dim vx2(5) As Single
Dim vy2(5) As Single
Dim vvx2 As Single
Dim vvy2 As Single
Dim gx As Single
Dim gy As Single
Dim gx2 As Single
Dim gy2 As Single
Dim xx As Single
Dim yy As Single
Dim xx2 As Single
```

```
Dim yy2 As Single
Dim f(5) As Single
Dim fbest As Single
Dim f2(5) As Single
Dim f2best As Single
Dim fi11 As Single
Dim fi21 As Single
Dim fi12 As Single
Dim fi22 As Single
Dim erro As Single
Dim erro2 As Single

Range(Cells(1, 1), Cells(100, 100)).delete

Range(Cells(1, 1), Cells(100, 100)).Interior.Color = vbWhite

ActiveSheet.Shapes.AddLine(10, 5, 10, 15).Select '----------------------------------------------(1)
ActiveSheet.Shapes.AddLine(5, 10, 15, 10).Select

fi11 = 0.5 '-------------------------------------------------------------------------------------(2)
fi21 = 0.5
fi12 = 0.5
fi22 = 0.5

fbest = 1000000 '--------------------------------------------------------------------------------(3)
f2best = 1000000

Set W = Worksheets("Plan1")

Randomize


For i = 1 To 5
```

```
x(i) = (100 * Rnd) '----------------------------------------------------------------------(4)
y(i) = (100 * Rnd)
x2(i) = (100 * Rnd)
y2(i) = (100 * Rnd)

vx(i) = (100 * Rnd) '----------------------------------------------------------------------(5)
vy(i) = (100 * Rnd)
vx2(i) = (100 * Rnd)
vy2(i) = (100 * Rnd)

f(i) = (x(i) - 10) ^ 2 + (y(i) - 10) ^ 2 '-------------------------------------------------(6)
f2(i) = (x2(i) - 10) ^ 2 + (y2(i) - 10) ^ 2

With W.Shapes.AddShape(msoShapeOval, x(i), y(i), 2, 2) '----------------------------------(7)
.Fill.ForeColor.RGB = RGB(0, 0, 255)
End With

With W.Shapes.AddShape(msoShapeOval, x2(i), y2(i), 2, 2)
.Fill.ForeColor.RGB = RGB(255, 0, 255)
.Line.ForeColor.SchemeColor = 14
End With

    xbest(i) = x(i) '----------------------------------------------------------------------(8)
    ybest(i) = y(i)

    x2best(i) = x2(i)
    y2best(i) = y2(i)

If f(i) < fbest Then '----------------------------------------------------------------------(9)
    fbest = f(i)
    gx = x(i)
    gy = y(i)
End If
If f2(i) < f2best Then
```

```
        f2best = f2(i)
        gx2 = x2(i)
        gy2 = y2(i)
    End If

Next i

k = 1

erro = 100000 '-------------------------------------------------------------------------------(10)
erro2 = 100000

Do While erro > 0.01 And erro2 > 0.01 '-----------------------------------------------------------(11)

    If k > 100 Then
        erro = 0
        erro2 = 0
    Else

        For i = 1 To 5

            vvx = vx(i) + (fi11 * Rnd()) * (xbest(i) - x(i)) + (fi21 * Rnd()) * (gx - x(i)) '-----(12)
            vvy = vy(i) + (fi11 * Rnd()) * (ybest(i) - y(i)) + (fi21 * Rnd()) * (gy - y(i))
            vvx2 = vx2(i) + (fi12 * Rnd()) * (x2best(i) - x2(i)) + (fi22 * Rnd()) * (gx2 - x2(i))
            vvy2 = vy2(i) + (fi12 * Rnd()) * (y2best(i) - y2(i)) + (fi22 * Rnd()) * (gy2 - y2(i))

            xx = vvx + x(i) '-------------------------------------------------------------------(13)
            yy = vvy + y(i)
            xx2 = vvx2 + x2(i)
            yy2 = vvy2 + y2(i)

            f(i) = (xx - 10) ^ 2 + (yy - 10) ^ 2 '-----------------------------------------------(14)
            f2(i) = (xx2 - 10) ^ 2 + (yy2 - 10) ^ 2
```

```
        xbest(i) = x(i) '----------------------------------------------------------------------(15)
        ybest(i) = y(i)
        x2best(i) = x2(i)
        y2best(i) = y2(i)

        If f(i) < fbest Then '-----------------------------------------------------------------(16)
            fbest = f(i)
            gx = x(i)
            gy = y(i)
        End If
        If f2(i) < f2best Then
            f2best = f2(i)
            gx2 = x2(i)
            gy2 = y2(i)
        End If

        x(i) = xx '-----------------------------------------------------------------------------(17)
        y(i) = yy
        x2(i) = xx2
        y2(i) = yy2

        vx(i) = vvx
        vy(i) = vvy
        vx2(i) = vvx2
        vy2(i) = vvy2
    Next i

End If

For Each pto In ActiveSheet.Shapes
    pto.delete
Next pto

For i = 1 To 5 '------------------------------------------------------------------------------(18)
```

```
        With W.Shapes.AddShape(msoShapeOval, x(i), y(i), 2, 2)
        .Fill.ForeColor.RGB = RGB(0, 0, 255)
        End With

        With W.Shapes.AddShape(msoShapeOval, x2(i), y2(i), 2, 2)
        .Fill.ForeColor.RGB = RGB(255, 0, 255)
        .Line.ForeColor.SchemeColor = 14
        End With
    Next i

    ActiveSheet.Shapes.AddLine(10, 5, 10, 15).Select '----------------------------------------(19)
    ActiveSheet.Shapes.AddLine(5, 10, 15, 10).Select

    Application.Wait (Now + TimeValue("0:00:01"))

    k = k + 1
    erro = Abs(fbest - 0) '-------------------------------------------------------------------(20)
    erro2 = Abs(f2best - 0)

Loop
End

End Sub
```

Anexo 3 – Programação no MATLAB do PSO de Uma População

```
tic %-------------------------------------------------------------(1)

clear all %-------------------------------------------------------(2)

fi1=0.5; %--------------------------------------------------------(3)
fi2=0.5;

fbest=100000; %---------------------------------------------------(4)

for i=1:10 %------------------------------------------------------(5)

    x(i)=(100*rand); %--------------------------------------------(6)
    y(i)=(100*rand);
    vx(i)=(100*rand);
    vy(i)=(100*rand);

    f(i)=(x(i)-40)^2+(y(i)-40)^2; %-------------------------------(7)

    px(i)=x(i); %-------------------------------------------------(8)
    py(i)=y(i);

    if f(i)<fbest %-----------------------------------------------(9)
        fbest=f(i);
        gx=x(i);
        gy=y(i);
    end

end

k=1; %------------------------------------------------------------(10)

erro=100000; %----------------------------------------------------(11)

while erro>0.01 %-------------------------------------------------(12)

    if k>100 %----------------------------------------------------(13)
        erro=0;
    else

        for i=1:10 %--------------------------------------------(14)

            vvx=vx(i)+(fi1*rand)*(px(i)-x(i))+(fi2*rand)*(gx-x(i)); %(15)
            vvy=vy(i)+(fi1*rand)*(py(i)-y(i))+(fi2*rand)*(gy-y(i));

            xx=x(i)+vvx; %------------------------------------(16)
            yy=y(i)+vvy;

            f(i)=(x(i)-40)^2+(y(i)-40)^2; %-------------------(17)

            px(i)=x(i); %-------------------------------------(18)
            py(i)=y(i);

            if f(i)<fbest %-----------------------------------(19)
```

```
                fbest=f(i);
                gx=x(i);
                gy=y(i);
            end

            x(i)=xx; %-------------------------------------------------(20)
            y(i)=yy;

            vx(i)=vvx; %-----------------------------------------------(21)
            vy(i)=vvy;

        end

    plot(x,y,'.', 'markersize',20) %---------------------------------(22)
    axis([-100 100 -100 100]) %--------------------------------------(23)
    hold on %--------------------------------------------------------(24)
    plot(gx,gy,'+r', 'markersize', 10) %-----------------------------(25)
    axis([-100 100 -100 100]) %--------------------------------------(26)
    pause(0.1) %-----------------------------------------------------(27)
    grid %-----------------------------------------------------------(28)
    hold off %-------------------------------------------------------(29)

    erro=abs(fbest-0); %---------------------------------------------(30)
    k=k+1; %---------------------------------------------------------(31)

    end

    pause %----------------------------------------------------------(32)

end

toc %------------------------------------------------------------------(33)
```

Anexo 4 – Programação no MATLAB do PSO de Duas Populações Independentes

```
tic

clear all

fi11=0.5; %-------------------------------------------------------------(1)
fi21=0.5;
fi12=0.5;
fi22=0.5;

f1best=100000; %--------------------------------------------------------(2)
f2best=100000;

for i=1:10

    x1(i)=(100*rand); %-------------------------------------------------(3)
    y1(i)=(100*rand);
    vx1(i)=(100*rand);
    vy1(i)=(100*rand);

    f1(i)=x1(i)^2+y1(i)^2; %--------------------------------------------(4)

    px1(i)=x1(i); %-----------------------------------------------------(5)
    py1(i)=y1(i);

    x2(i)=(100*rand); %-------------------------------------------------(6)
    y2(i)=(100*rand);
    vx2(i)=(100*rand);
    vy2(i)=(100*rand);

    f2(i)=(x2(i)-40)^2+(y2(i)-40)^2; %----------------------------------(7)

    px2(i)=x2(i);
    py2(i)=y2(i);

    if f1(i)<f1best %---------------------------------------------------(8)
        f1best=f1(i);
        gx1=x1(i);
        gy1=y1(i);
    end

    if f2(i)<f2best %---------------------------------------------------(9)
        f2best=f2(i);
        gx2=x2(i);
        gy2=y2(i);
    end
end

k=1;
```

```
erro1=100000; %-----------------------------------------------------(10)
erro2=100000;

while (erro1>0.01)&&(erro2>0.01) %-----------------------------------(11)

    if k>100 %-----------------------------------------------------(12)
        erro1=0;
        erro2=0;
    else
        for i=1:10 %-----------------------------------------------(13)

            vvx1=vx1(i)+(fi11*rand)*(px1(i)-x1(i))+(fi12*rand)*(gx1-x1(i));
%(14)
            vvy1=vy1(i)+(fi11*rand)*(py1(i)-y1(i))+(fi12*rand)*(gy1-y1(i));
            xx1(i)=x1(i)+vvx1;
            yy1(i)=y1(i)+vvy1;

            f1(i)=x1(i)^2+y1(i)^2;

            vvx2=vx2(i)+(fi12*rand)*(px2(i)-x2(i))+(fi22*rand)*(gx2-x2(i));
%(15)
            vvy2=vy2(i)+(fi12*rand)*(py2(i)-y2(i))+(fi22*rand)*(gy2-y2(i));
            xx2(i)=x2(i)+vvx2;
            yy2(i)=y2(i)+vvy2;

            f2(i)=(x2(i)-40)^2+(y2(i)-40)^2;

            px1(i)=x1(i); %---------------------------------------(16)
            py1(i)=y1(i);

            px2(i)=x2(i);
            py2(i)=y2(i);

        if f1(i)<f1best %--------------------------------------------(17)
            f1best=f1(i);
            gx1=x1(i);
            gy1=y1(i);
        end

        x1(i)=xx1(i); %----------------------------------------------(18)
        y1(i)=yy1(i);
        vx1(i)=vvx1;
        vy1(i)=vvy1;

        if f2(i)<f2best %--------------------------------------------(19)
            f2best=f2(i);
            gx2=x2(i);
            gy2=y2(i);
        end

        x2(i)=xx2(i); %----------------------------------------------(20)
        y2(i)=yy2(i);
        vx2(i)=vvx2;
        vy2(i)=vvy2;

        end

    plot(x1(:),y1(:),'.b', 'markersize',20) %------------------------(21)
    axis([-100 100 -100 100])
```

```
    hold on
    plot(x2(:),y2(:),'.r', 'markersize',20) %-------------------------(22)
    plot(gx1,gy1,'+r', 'markersize', 10) %-----------------------------(23)
    plot(gx2,gy2,'+k', 'markersize', 10) %-----------------------------(24)
    axis([-100 100 -100 100])
    pause(0.1)
    grid
    hold off

    erro1=abs(f1best-0); %--------------------------------------------(25)
    erro2=abs(f2best-0);

    k=k+1;

    end

    pause

end

toc
```

Anexo 5 – Programação no MATLAB do PSO de Duas Populações Interagindo Entre Si

```
tic

clear all

fi11=0.5;
fi12=0.5;
fi21=0.5;
fi22=0.5;

f1best=100000;
f2best=100000;

for i=1:10

    x1(i)=(100*rand);
    y1(i)=(100*rand);
    vx1(i)=(100*rand);
    vy1(i)=(100*rand);

    px1(i)=x1(i);
    py1(i)=y1(i);

    x2(i)=(100*rand);
    y2(i)=(100*rand);
    vx2(i)=(100*rand);
    vy2(i)=(100*rand);

    px2(i)=x2(i);
    py2(i)=y2(i);

    f1(i)=x1(i)^2+y1(i)^2; %-----------------------------------------(1)
    f2(i)=x2(i)^2+y2(i)^2;

    if f1(i)<f1best
        f1best=f1(i);
        gx1=x1(i);
        gy1=y1(i);
    end

    if f2(i)<f2best
        f2best=f2(i);
        gx2=x2(i);
        gy2=y2(i);
    end

end

if f1best<f2best %-----------------------------------------------------(2)
    fi21=0.3;
```

```
    fi22=0.3;
else
    fi11=0.3;
    fi12=0.3;
end

k=1;

erro1=100000;
erro2=100000;

while (erro1>0.1)&&(erro2>0.1)

    if k>100
        erro1=0;
        erro2=0;
    else
        for i=1:10

            vvx1=vx1(i)+(fi11*rand)*(px1(i)-x1(i))+(fi12*rand)*(gx1-x1(i));
            vvy1=vy1(i)+(fi11*rand)*(py1(i)-y1(i))+(fi12*rand)*(gy1-y1(i));
            xx1(i)=x1(i)+vvx1;
            yy1(i)=y1(i)+vvy1;

            f1(i)=x1(i)^2+y1(i)^2;

            vvx2=vx2(i)+(fi21*rand)*(px2(i)-x2(i))+(fi22*rand)*(gx2-x2(i));
            vvy2=vy2(i)+(fi21*rand)*(py2(i)-y2(i))+(fi22*rand)*(gy2-y2(i));
            xx2(i)=x2(i)+vvx2;
            yy2(i)=y2(i)+vvy2;

            f2(i)=x2(i)^2+y2(i)^2;

            px1(i)=x1(i);
            py1(i)=y1(i);

            px2(i)=x2(i);
            py2(i)=y2(i);

        if f1(i)<f1best
            f1best=f1(i);
            gx1=x1(i);
            gy1=y1(i);
        end

        x1(i)=xx1(i);
        y1(i)=yy1(i);
        vx1(i)=vvx1;
        vy1(i)=vvy1;

        if f2(i)<f2best
            f2best=f2(i);
            gx2=x2(i);
            gy2=y2(i);
        end

        x2(i)=xx2(i);
        y2(i)=yy2(i);
        vx2(i)=vvx2;
```

```
        vy2(i)=vvy2;

        end

    if f1best<f2best %-------------------------------------------------(3)
        fi21=0.3;
        fi22=0.3;
    else
        fi11=0.3;
        fi22=0.3;
    end

    plot(x1(:),y1(:),'.b','markersize',20)
    axis([-100 100 -100 100])
    hold on
    plot(x2(:),y2(:),'.r','markersize',20)
    plot(gx1,gy1,'+r','markersize',10)
    plot(gx2,gy2,'+k','markersize',10)
    axis([-100 100 -100 100])
    pause(0.1)
    grid
    hold off

    erro1=abs(f1best-0);
    erro2=abs(f2best-0);

    k=k+1;

    end

    pause

end

toc
```

Anexo 6 – Programação no MATLAB do PSO de Uma População Com Restrição

```
tic

clear all

b1=0; %-------------------------------------------------(1)
b2=-100;

xl10=200; %-------------------------------------------(2)
yl10=400;
xl11=-200;
yl11=-400;

a=(yl11-yl10)/(xl11-xl10); %---------------------------(3)

xl20=xl10; %-------------------------------------------(4)
yl20=xl20*a+b2;
xl21=xl11;
yl21=xl21*a+b2;

fi1=0.5;
fi2=0.5;

fbest=100000000;

for i=1:10
    x(i)=(100*rand); %-------------------------------(5)
    y(i)=(100*rand);

    while (x(i)<(y(i)-b1)/a)||(x(i)>(y(i)-b2)/a) %-------(6)
    x(i)=(100*rand); %-------------------------------(7)
    end
    while (y(i)>(x(i)*a+b1))||(y(i)<(x(i)*a+b2)) %-------(8)
    y(i)=(100*rand); %-------------------------------(9)
    end

    vx(i)=(100*rand); %-------------------------------(10)
    vy(i)=(100*rand);

    f(i)=(x(i)+125)^2+(y(i)+300)^2; %-----------------(11)

    px(i)=x(i);
    py(i)=y(i);

    if f(i)<fbest
        fbest=f(i);
        gx=x(i);
        gy=y(i);
    end
end
```

```
k=1;
erro=100000;

while erro>0.01

    if k>100
        erro=0;
    else
        for i=1:10
            vvx=vx(i)+(fi1*rand)*(px(i)-x(i))+(fi2*rand)*(gx-x(i));
            vvy=vy(i)+(fi1*rand)*(py(i)-y(i))+(fi2*rand)*(gy-y(i));
            xx=x(i)+vvx;
            yy=y(i)+vvy;

            f(i)=(x(i)+125)^2+(y(i)+300)^2;

            px(i)=x(i);
            py(i)=y(i);

        if f(i)<fbest
            fbest=f(i);
            gx=x(i);
            gy=y(i);
        end

        if (xx>(yy-b1)/a) %------------------------------(12)
            x(i)=xx;
        else
            xx=(yy-b1)/a;
            x(i)=xx;
        end

        if (xx<(yy-b2)/a) %------------------------------(13)
            x(i)=xx;
        else
            xx=(yy-b2)/a;
            x(i)=xx;
        end

        if (yy<(xx*a+b1)) %------------------------------(14)
            y(i)=yy;
        else
            yy=xx*a+b1;
            y(i)=yy;
        end

        if (yy>(xx*a+b2)) %------------------------------(15)
            y(i)=yy;
        else
            yy=xx*a+b2;
            y(i)=yy;
        end

        vx(i)=vvx;
        vy(i)=vvy;
    end

    plot(x,y,'.', 'markersize', 20)
```

```
    axis([xl11 xl10 yl11 yl10]) %------------------------(16)
    hold on
    line([xl11 xl10],[yl11 yl10],[1 1],'LineStyle','-') %(17)
    line([xl21 xl20],[yl21 yl20],[1 1],'LineStyle','-') %(18)
    plot(-125,-300,'+r', 'markersize', 10) %-------------(19)
    axis([xl11 xl10 yl11 yl10])
    pause(0.1)
    grid
    hold off

    erro=abs(fbest-0);
    k=k+1;

    end

    pause

end

toc
```

Anexo 7 – Banco de Dados para o Gráfico da Figura 3.7

k	*erro*
1	74,6691
2	74,6691
3	74,6691
4	74,6691
5	74,6691
6	74,6691
7	13,3342
8	13,3342
9	13,3342
10	13,3342
11	13,3342
12	12,0782
13	12,0782
14	12,0782
15	12,0782
16	12,0782
17	1,9109
18	1,9109
19	1,0630
20	1,0630
21	1,0630
22	1,0630
23	1,0630
24	0,5702
25	0,5702
26	0,5702
27	0,5168
28	0,1565
29	0,1565
30	0,0182
31	0,0182
32	0,0182
33	0,0182
34	0,0182
35	0,0182
36	0,0029

Anexo 8 – Banco de Dados para o Gráfico da Figura 3.14

fi1=fi2=0,5	*fi1=fi2=0,9*	*fi1=fi2=0,1*
4,360311	3,010494	12,919984
4,897896	1,761470	13,145536
4,930196	3,156171	13,199416
4,643559	3,546678	12,952193
5,947234	3,159599	13,194230
5,012071	3,146966	13,033891
6,097910	2,900780	13,145334
5,446977	3,262794	13,232959
5,536429	3,526360	13,054464
5,120322	3,655215	13,170505

Anexo 9 – Banco de Dados para o Gráfico da Figura 4.1

Data	Volume	Volume
2/6/2008	0,171671	0,038735
3/6/2008	0,214704	0,034511
4/6/2008	0,280599	0,020975
5/6/2008	0,202192	0,020993
6/6/2008	0,261286	0,03443
9/6/2008	0,130232	0,039744
10/6/2008	0,181551	0,032717
11/6/2008	0,173173	0,03309
12/6/2008	0,218642	0,022978
13/6/2008	0,158561	0,014914
16/6/2008	0,255186	0,028577
17/6/2008	0,171184	0,033264
18/6/2008	0,101483	0,030867
19/6/2008	0,226004	0,027453
20/6/2008	0,167694	0,021517
23/6/2008	0,156031	0,02521
24/6/2008	0,143939	0,029368
25/6/2008	0,189638	0,03163
26/6/2008	0,126988	0,019452
27/6/2008	0,132858	0,011483
30/6/2008	0,177931	0,014086
1/7/2008	0,141271	0,020078
2/7/2008	0,167389	0,023315
3/7/2008	0,156002	0,017106
4/7/2008	0,162867	0,018508
7/7/2008	0,178778	0,02606
8/7/2008	0,209636	0,019568
10/7/2008	0,193926	0,022623
11/7/2008	0,187293	0,031903
14/7/2008	0,130588	0,028246
15/7/2008	0,145126	0,042531
16/7/2008	0,135061	0,081301
17/7/2008	0,162629	0,044586
18/7/2008	0,126846	0,07558
21/7/2008	0,157121	0,035297
22/7/2008	0,14176	0,04211
23/7/2008	0,169403	0,025359
24/7/2008	0,15482	0,060969
25/7/2008	0,146088	0,037934
28/7/2008	0,156065	0,055737
29/7/2008	0,154421	0,044655
30/7/2008	0,16704	0,028563
31/7/2008	0,120948	0,033846
1/8/2008	0,128556	0,040247
4/8/2008	0,153963	0,026455

5/8/2008	0,150131	0,028298
6/8/2008	0,128874	0,018559
7/8/2008	0,183199	0,036926
8/8/2008	0,145179	0,030871
11/8/2008	0,112837	0,038455
12/8/2008	0,141675	0,028289
13/8/2008	0,105088	0,021089
14/8/2008	0,142114	0,032701
15/8/2008	0,130165	0,030182
18/8/2008	0,177512	0,018093
19/8/2008	0,169799	0,031627
20/8/2008	0,176504	0,036001
21/8/2008	0,159581	0,029926
22/8/2008	0,165779	0,041234
25/8/2008	0,197749	0,033106
26/8/2008	0,172365	0,021856
27/8/2008	0,150955	0,034438
28/8/2008	0,133851	0,039825
29/8/2008	0,118696	0,023602
1/9/2008	0,147538	0,024743
2/9/2008	0,163851	0,028537
3/9/2008	0,13535	0,034909
4/9/2008	0,130234	0,025153
5/9/2008	0,148299	0,025973
8/9/2008	0,159419	0,038292
9/9/2008	0,164286	0,03514
10/9/2008	0,258119	0,031401
11/9/2008	0,210767	0,026528
12/9/2008	0,188941	0,026126
15/9/2008	0,22331	0,020712
16/9/2008	0,170727	0,024623
17/9/2008	0,155157	0,02796
18/9/2008	0,182166	0,023378
19/9/2008	0,169428	0,028313
22/9/2008	0,210583	0,036403
23/9/2008	0,198326	0,040183
24/9/2008	0,197095	0,033282
25/9/2008	0,146293	0,041672
26/9/2008	0,142245	0,061593
29/9/2008	0,189731	0,056302
30/9/2008	0,159557	0,037914
1/10/2008	0,184166	0,038643
2/10/2008	0,164201	0,063153
3/10/2008	0,174777	0,031555
6/10/2008	0,203381	0,036145
7/10/2008	0,193849	0,046067
8/10/2008	0,211723	0,045057
9/10/2008	0,220921	0,039608
10/10/2008	0,224282	0,037102
13/10/2008	0,20429	0,041056
14/10/2008	0,220132	0,057206
15/10/2008	0,181332	0,053871

16/10/2008	0,230065	0,039052
17/10/2008	0,203196	0,047881
20/10/2008	0,227985	0,031975
21/10/2008	0,187675	0,048287
22/10/2008	0,185753	0,040889
23/10/2008	0,190276	0,04272
24/10/2008	0,194428	0,040282
27/10/2008	0,200739	0,030516
28/10/2008	0,207184	0,040372
29/10/2008	0,212148	0,030811
30/10/2008	0,180284	0,032219
31/10/2008	0,168729	0,064319
3/11/2008	0,121917	0,049145
4/11/2008	0,194628	0,03262
5/11/2008	0,165172	0,046341
6/11/2008	0,19312	0,053251
7/11/2008	0,198034	0,040678
10/11/2008	0,207485	0,043398
11/11/2008	0,190243	0,039606
12/11/2008	0,262627	0,03069
13/11/2008	0,216623	0,026937
14/11/2008	0,191824	0,025072
17/11/2008	0,215076	0,025252
18/11/2008	0,216908	0,024851
19/11/2008	0,186846	0,025881
21/11/2008	0,205388	0,027648
24/11/2008	0,227573	0,030032
25/11/2008	0,225555	0,023743
26/11/2008	0,188644	0,018887
27/11/2008	0,212425	0,025375
28/11/2008	0,186812	0,024433
1/12/2008	0,186032	0,025094
2/12/2008	0,167402	0,025705
3/12/2008	0,226138	0,023282
4/12/2008	0,18504	0,022415
5/12/2008	0,190534	0,040107
8/12/2008	0,187059	0,043477
9/12/2008	0,173571	0,030989
10/12/2008	0,230935	0,047428
11/12/2008	0,273348	0,029884
12/12/2008	0,248768	0,047892
15/12/2008	0,234611	0,02848
16/12/2008	0,195468	0,039143
17/12/2008	0,147569	0,036048
18/12/2008	0,208148	0,033011
19/12/2008	0,198321	0,032352
22/12/2008	0,190434	0,02302
23/12/2008	0,163633	0,029794
26/12/2008	0,21857	0,028023
29/12/2008	0,178179	0,027795
30/12/2008	0,148773	0,021585
2/1/2009	0,2002	0,049393

5/1/2009	0,173634	0,033278
6/1/2009	0,159542	0,033185
7/1/2009	0,185864	0,033222
8/1/2009	0,125246	0,02613
9/1/2009	0,141189	0,043993
12/1/2009	0,194752	0,042104
13/1/2009	0,181178	0,027966
14/1/2009	0,160241	0,051247
15/1/2009	0,204346	0,041274
16/1/2009	0,184118	0,028418
19/1/2009	0,205162	0,018148
20/1/2009	0,152998	0,034529
21/1/2009	0,162621	0,028413
22/1/2009	0,177059	0,041152
23/1/2009	0,188023	0,054178
26/1/2009	0,203298	0,038348
27/1/2009	0,147671	0,07688
28/1/2009	0,196114	0,048044
29/1/2009	0,150035	0,036057
30/1/2009	0,188857	0,036343
2/2/2009	0,186377	0,049533
3/2/2009	0,164584	0,075117
4/2/2009	0,170631	0,073762
5/2/2009	0,178509	0,0905
6/2/2009	0,163262	0,096406
9/2/2009	0,230177	0,069369
10/2/2009	0,185006	0,061158
11/2/2009	0,207315	0,048566
12/2/2009	0,179963	0,045051
13/2/2009	0,193032	0,031058
16/2/2009	0,174129	0,020322
17/2/2009	0,162939	0,053939
18/2/2009	0,179371	0,037938
19/2/2009	0,19823	0,024679
20/2/2009	0,167414	0,091039
25/2/2009	0,168892	0,067536
26/2/2009	0,189522	0,037369
27/2/2009	0,211394	0,045573
2/3/2009	0,191234	0,055362
3/3/2009	0,209569	0,044137
4/3/2009	0,234603	0,048785