

Trabalho 1
Aprendizado de Máquina - 2018.1
Universidade Federal do Rio de Janeiro

Raphael Barros Parreira

20 de Julho de 2018



Índice

| | | |
|----------|--|-----------|
| 1 | Introdução | 3 |
| 1.1 | Arquivos fornecidos pelo Kaggle | 3 |
| 1.2 | Como funciona a submissão à competição | 3 |
| 1.3 | Premissas | 3 |
| 2 | Pré-processamento dos dados | 4 |
| 2.1 | Tratamento de Outliers | 4 |
| 2.2 | Variável Alvo | 5 |
| 3 | Trechos importantes do código | 7 |
| 3.1 | Variáveis importantes | 7 |
| 3.2 | Botão de inserção | 7 |
| 3.3 | Loop principal | 8 |
| 3.4 | Explicação do Programa | 10 |
| 4 | Exemplos de simulações | 11 |
| 4.1 | Teste 1 | 11 |
| 4.1.1 | Configurações | 11 |
| 4.1.2 | Resultado da Simulação | 11 |
| 4.2 | Teste 2 | 12 |
| 4.2.1 | Configurações | 12 |
| 4.2.2 | Resultado da Simulação | 12 |
| 4.3 | Teste 3 | 13 |
| 4.3.1 | Configurações | 13 |
| 4.3.2 | Resultado da Simulação | 13 |
| 4.4 | Teste 4 | 14 |
| 4.4.1 | Configurações | 14 |
| 4.4.2 | Resultado da Simulação | 15 |
| 5 | Anexo | 15 |

1 Introdução

O trabalho consiste na participação de uma competição do Kaggle. A competição prevê a predição de preços de casas baseada em suas características. Para isso, é necessário que seja aplicado o conhecimento adquirido no curso para fazer com que um programa de computador seja capaz de aprender como fazer essa predição baseado numa base de dados conhecida previamente.

Além de fornecer a base de dados de treino (que contém os preços das casas) e a base de teste (base usada para fazer a predição final), o Kaggle também fornece materiais de outros participantes, que auxiliam o processo de aprendizagem e desenvolvimento do programa.

Uma outra informação presente na competição é a descrição de cada uma das características possíveis das casas, para que auxilie o desenvolvedor a escolher o que fazer com cada uma das colunas da base de dados.

1.1 Arquivos fornecidos pelo Kaggle

- *train.csv* - dataset de treino
- *test.csv* - dataset de teste
- *data_description.txt* - descrição de cada uma das características da casa
- *sample_submission.csv* - exemplo de arquivo para ser submetido ao Kaggle

1.2 Como funciona a submissão à competição

Para submeter a resposta ao Kaggle, o competidor deve criar um arquivo *.csv* (valores separados por vírgulas) contendo duas colunas: Id e SalePrice. Além disso, os preços e os Ids devem corresponder, para que o sistema do Kaggle consiga validar o preço para o seu respectivo Id.

Ao enviar o arquivo para a validação, o Kaggle verifica a pontuação da tentativa e informa a colocação.

1.3 Premissas

O trabalho apresenta, como premissa, o fato de se usar apenas algoritmos de regressão para realizar a predição dos preços. Além disso, deixou livre a escolha dos algoritmos por parte do participante. Sendo assim, os algoritmos escolhidos foram: Ridge, Lasso, ElasticNet e KernelRidge.

2 Pré-processamento dos dados

Para que os algoritmos tenham uma maior eficácia, é necessário que os dados fornecidos pelo desafio sejam processados. Esse processamento é responsável por excluir dados que prejudiquem o aprendizado, preencher os faltantes (quando possível), criação de novas colunas, compilando e/ou combinando as características já existentes e, se necessário, manipulação dos dados para simplificação de categorias, com o objetivo de tornar o conjunto de dados mais enxuto.

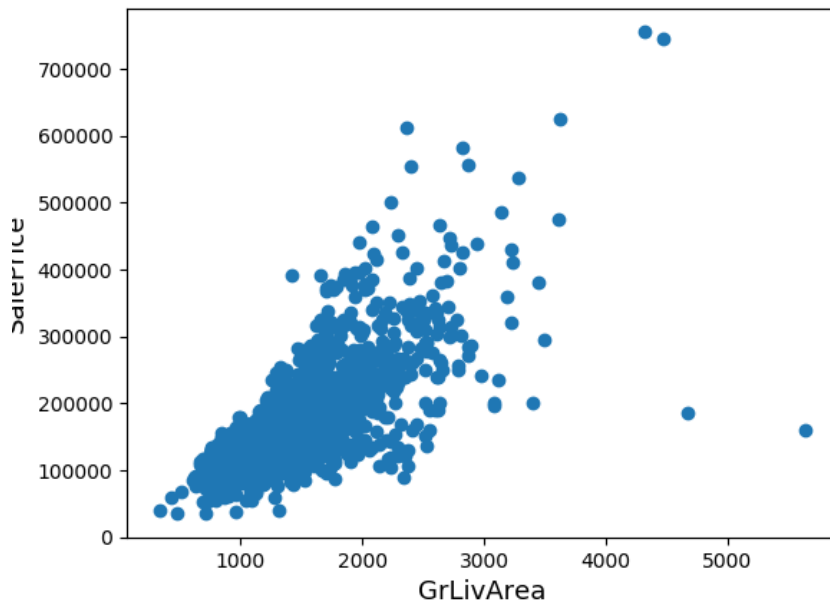
2.1 Tratamento de Outliers

Um dos passos importantes para o tratamento de dados, é a identificação e remoção de pontos muito fora do padrão, para que o algoritmo não se atrapalhe com pontos que não fazem parte do conjunto de dados normalmente, que tem pouca ocorrência.

No caso desse desafio, o próprio autor do dataset indica que existem outliers no conjunto de dados de treino.

Podemos observar os outliers no gráfico abaixo.

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3
4 df_train=pd.read_csv('./input/train.csv')
5
6 fig, ax = plt.subplots()
7 ax.scatter(x = df_train['GrLivArea'], y = df_train['SalePrice'])
8 plt.ylabel('SalePrice', fontsize=13)
9 plt.xlabel('GrLivArea', fontsize=13)
10 plt.show()
```



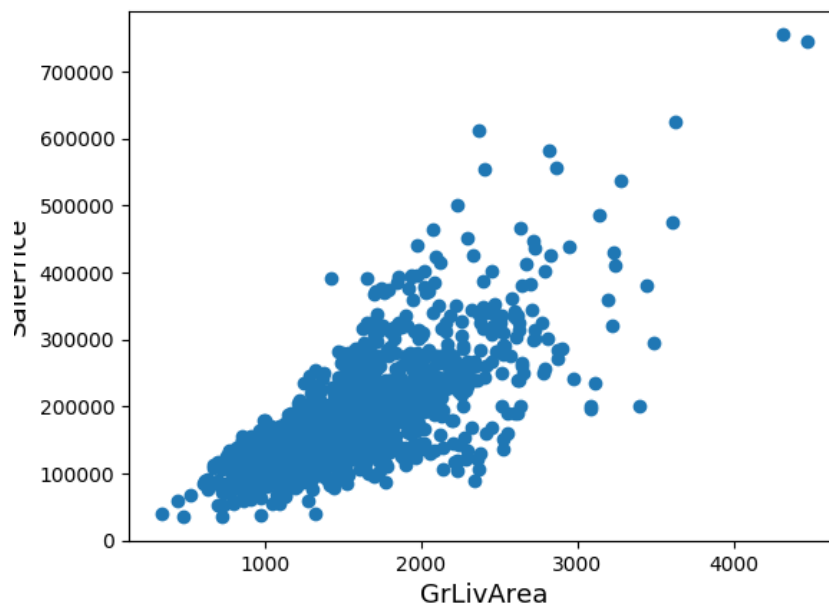
O autor recomenda remover os dois pontos que tem uma grande área, e

que foram vendidos por um preço muito baixo. Que são os dois pontos com $GrLivArea > 4.000$ & $SalePrice < 300.000$.

```

1 df_train = df_train.drop(df_train[(df_train['GrLivArea'] > 4000) & (df_train['
  SalePrice'] < 300000)].index)
2
3 fig, ax = plt.subplots()
4 ax.scatter(x = df_train['GrLivArea'], y = df_train['SalePrice'])
5 plt.ylabel('SalePrice', fontsize=13)
6 plt.xlabel('GrLivArea', fontsize=13)
7 plt.show()

```



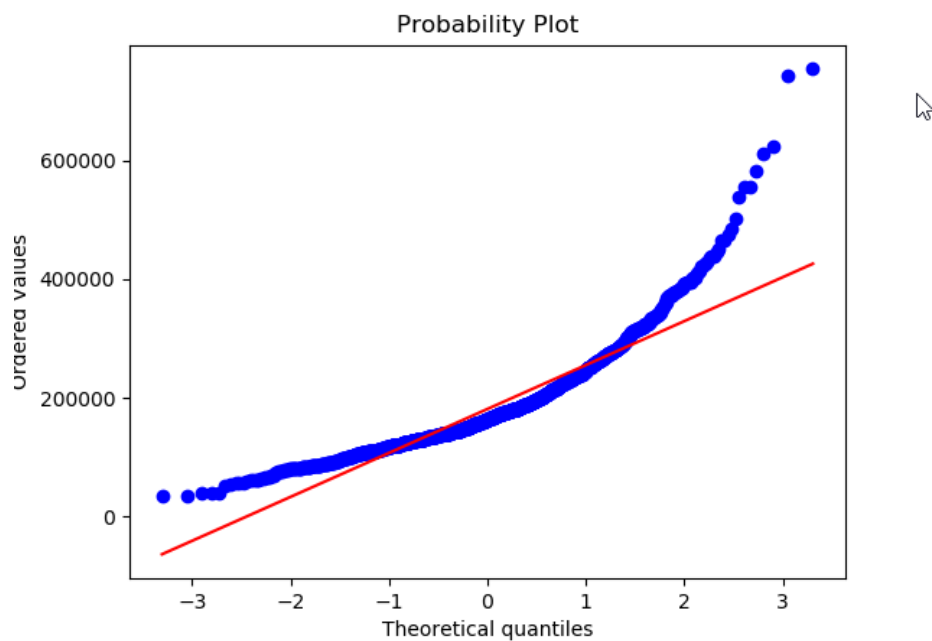
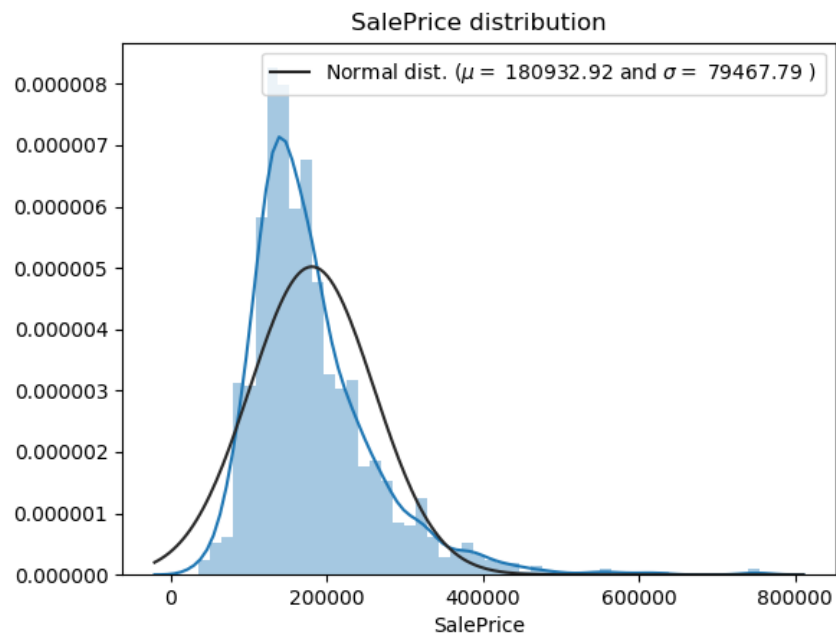
2.2 Variável Alvo

Como dito anteriormente, a variável a ser calculada é o preço do imóvel. Sendo assim, é necessário avaliar a relação do SalePrice com as outras variáveis do conjunto de dados, e também a sua normalização.

```

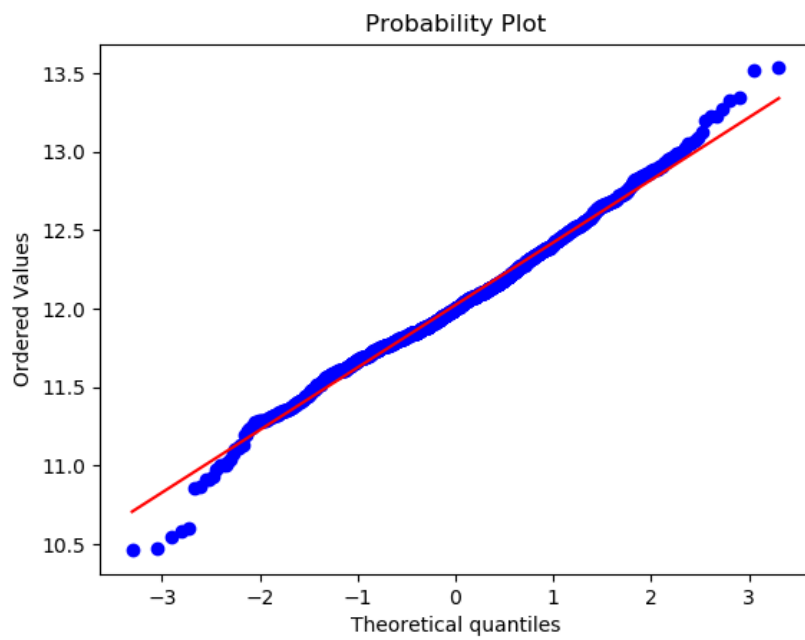
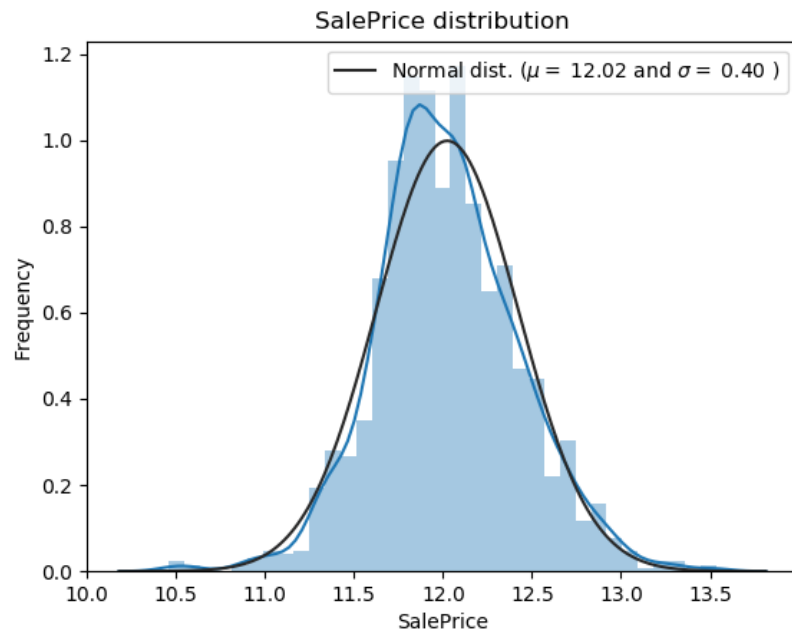
1 sns.distplot(df_train['SalePrice'], fit=norm);
2
3 # Get the fitted parameters used by the function
4 (mu, sigma) = norm.fit(df_train['SalePrice'])
5 print('\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
6
7 #Now plot the distribution
8 plt.legend(['Normal dist. ($\mu$ {:.2f} and $\sigma$ {:.2f} )'.format(mu, sigma)],
9           loc='best')
10 plt.ylabel('Frequency')
11 plt.title('SalePrice distribution')
12
13 #Get also the QQ-plot
14 fig = plt.figure()
15 res = stats.probplot(df_train['SalePrice'], plot=plt)
16 plt.show()

```



Como podemos perceber, a variável é *skewed* positivamente. Com isso, para que os regressores lineares funcionem de uma melhor forma, é importante que a variável seja normalizada.

```
1 df_train["SalePrice"] = np.log1p(df_train["SalePrice"])
```



3 Trechos importantes do código

Primeiramente é importante ressaltar que os trechos de códigos vistos abaixo foram escritos na linguagem do Matlab.

3.1 Variáveis importantes

```
1 handles.ns = 0;
2 handles.Soriginal = [];
3
4 .
5 .
6 .
7
8 chegada = str2num(get(handles.edit1, 'String'));
9
10 tempComp = str2num(get(handles.edit2, 'String'));
11
12 deadline = str2num(get(handles.edit3, 'String'));
13
14 prioridade = str2num(get(handles.edit4, 'String'));
```

No trecho acima pode-se observar algumas variáveis importantes para o bom funcionamento do protocolo estudado, em que:

- `handles.ns` é o número de semáforos utilizados por um determinado conjunto de tarefas;
- `handles.Soriginal` são os parâmetros de cada semáforo e sua relação com cada tarefa, ou seja, quanto tempo a tarefa irá utilizá-lo e a partir de que instante ela necessitará desse semáforo.

São as variáveis globais do código, isto é, podem ser acessadas por qualquer função do programa. Já as variáveis abaixo, são as variáveis locais da função principal do programa.

- `chegada` é um array que contém os tempo de chegada de cada tarefa;
- `tempComp` é um array que armazena os tempos de execução para cada tarefa;
- `deadline` array para armazenar os tempos limites (deadlines) de cada tarefa a ser executada;
- `prioridade` é um array que armazena as prioridades "normal" de cada tarefa;

É importante observar a função `get(handle, '')` vistas na atribuição de cada uma dessas últimas variáveis listadas, isso uma vez que esta função é responsável por adquirir o valor do campo correspondente àquela variável na interface.

3.2 Botão de inserção

Nesta seção analisa-se o funcionamento do botão *Inserir*.

```
1 function pushbutton2_Callback(hObject, eventdata, handles)
2 w = handles.ns + 1;
3 initial_temp = cellstr(get(handles.listbox1, 'String'));
4 a = get(handles.edit6, 'String');
5 temp = [initial_temp; a];
6 set(handles.listbox1, 'String', temp);
7 b = str2num(a);
8 handles.ns = w;
9 handles.Soriginal(:,w) = str2num(get(handles.edit6, 'String'));
10 guidata(hObject, handles);
```


Este botão, como o nome sugere serve para a inserção de um semáforo na execução das tarefas, para isso é necessário que se escreva os parâmetros do semáforo, isto é, tempo em que cada tarefa faz o requerimento para utilizá-lo; duração da execução de cada tarefa utilizando este semáforo e por fim o "status" da tarefa em relação ao semáforo sendo:

1. 0 \rightarrow para tarefas inativas;
2. 1 \rightarrow para tarefas ativas;
3. 2 \rightarrow para tarefas finalizadas ou inexistentes em relação ao semáforo.

3.3 Loop principal

Nesta seção é mostrado o trecho do código onde "a mágica acontece", isso tendo em vista que é nesse trecho que todas as comparações e definições de prioridades ocorrem como pode ser visto abaixo:

```

1  if length(tam)< 3;
2      tam(3) = 1;
3  end
4  for i=1:tam(3)
5      prioridadeBloqueada(i)=101;
6
7      [cores(i,1),cores(i,2),cores(i,3)]=HSVtoRGB(1+(i-1)*(358/(tam(3))),.41,.86);
8
9      for j=1:tam(1)
10         if S(j,2,i)>0
11             C(i)=j;
12             break;
13         end
14     end
15 end
16
17 C(length(C)+1)=100;
18
19 tempo=0;
20
21 for i=1:length(chegada)
22     prioridadeAtiva(i)=100;
23 end
24
25 ativos=zeros(length(chegada));
26
27
28 largura=max((sum(sum(S(:,2,:)))+sum(tempComp))*1.2,(max(deadline)+2));
29
30
31 tic;
32
33
34
35 cla
36 hold on
37 grid on
38 set(gca,'xtick',[0:1:largura])
39
40 xlim([0, largura])
41 ylim([0,20*(1+length(chegada))])
42 set(gca,'YTick',[0:20:20*length(chegada)])
43
44 for i=1:length(chegada)
45     plot([chegada(i),chegada(i)],[20*(length(chegada)+1-i),20*(length(chegada)+1-i)
46         +15], '-.b', 'LineWidth',2)
47     plot([deadline(i),deadline(i)],[20*(length(chegada)+1-i),20*(length(chegada)+1-i)
48         +15], '-.r', 'LineWidth',2)
49 end
50
51 drawnow
52
53 SmaiorCeiling=length(C);
54
55 while (true)
56     paraContador=toc;
57
58     tic;
59
60     tempo=tempo+paraContador;
61
62
63
64

```

```

65     for i=1:length(chegada)
66         if ~ativos(i) && chegada(i)<=tempo
67             ativos(i)=1;
68             prioridadeAtiva(i)=prioridade(i);
69         end
70     end
71
72     [minimo, proximoProcesso]=min(prioridadeAtiva);
73
74     for i=1:tam(3)
75         if S(proximoProcesso,3,i)==0 & compFeita(proximoProcesso) >= S(
76             proximoProcesso,1,i)
77             for j=1:tam(3)
78                 if any(S(:,3,j)==1) & C(SmaiorCeiling)>C(j)
79                     SmaiorCeiling=j;
80                 end
81             end
82             if prioridadeAtiva(proximoProcesso)<C(SmaiorCeiling) | any(S(
83                 proximoProcesso,3,:)==1)
84                 S(proximoProcesso,3,i)=1;
85             else
86                 prioridadeAtiva(find((S(:,3,SmaiorCeiling))==1))=proximoProcesso
87                 -0.1;
88                 prioridadeBloqueada(SmaiorCeiling)=proximoProcesso-0.1;
89             end
90         end
91     end
92 end
93
94 if minimo<100
95     if ~(any(S(proximoProcesso,3,:)==1))
96         compRestante(proximoProcesso)=compRestante(proximoProcesso)-paraContador
97         ;
98         if compRestante(proximoProcesso)<0
99             compRestante(proximoProcesso)=0;
100             tempo=tempo+compRestante(proximoProcesso);
101             paraContador=paraContador-compRestante(proximoProcesso);
102         end
103         compFeita(proximoProcesso)=compFeita(proximoProcesso)+paraContador;
104
105         rectangle('Position',[tempo-paraContador,20*(length(chegada)+1-
106             proximoProcesso),paraContador,10],'FaceColor',[0.6 0.6 0.9],'LineStyle','None')
107         ;
108         drawnow
109     else
110         zonaAtiva=[0 0];
111         for i=1:tam(3)
112             if S(proximoProcesso,3,i)==1 & S(proximoProcesso,1,i)>=zonaAtiva(2)
113                 zonaAtiva=[i S(proximoProcesso,1,i)];
114             end
115         end
116         S(proximoProcesso,2,zonaAtiva(1))=S(proximoProcesso,2,zonaAtiva(1))-
117         paraContador;
118         compFeita(proximoProcesso)=compFeita(proximoProcesso)+paraContador;
119
120         if S(proximoProcesso,2,zonaAtiva(1))<0
121             S(proximoProcesso,2,zonaAtiva(1))=0;
122             tempo=tempo+S(proximoProcesso,2,zonaAtiva(1));
123             paraContador=paraContador-S(proximoProcesso,2,zonaAtiva(1));
124         end
125         rectangle('Position',[tempo-paraContador,20*(length(chegada)+1-
126             proximoProcesso),paraContador,10],'FaceColor',cores(zonaAtiva(1,:)), 'LineStyle'
127             , 'None');
128         xlim([0,largura])
129         ylim([0,20*(1+length(chegada))])
130         set(gca,'YTick',[0:20:20*length(chegada)])
131         drawnow
132
133         if S(proximoProcesso,2,zonaAtiva(1))<=0
134             %Marca fim da zona
135             S(proximoProcesso,3,zonaAtiva(1))=2;
136             prioridadeBloqueada(zonaAtiva(1))=101;
137
138             prioridadeAtiva(proximoProcesso)=proximoProcesso;
139             for i=1:tam(3)
140                 if S(proximoProcesso,3,i)==1
141                     prioridadeAtiva(proximoProcesso)=min(prioridadeAtiva(
142                         proximoProcesso),prioridadeBloqueada(i));
143                 end
144             end
145             SmaiorCeiling=length(C);
146
147             for j=1:tam(3)
148                 if any(S(:,3,j)==1) & C(SmaiorCeiling)>C(j)
149                     SmaiorCeiling=j;
150                 end
151             end
152         end
153     end

```

```

154         end
155     end
156 end
157
158     end
159 end
160
161     end
162     if compRestante(proximoProcesso) <= 0
163         prioridadeAtiva(proximoProcesso) = 105;
164         if max(compRestante) <= 0
165             break;
166         end
167     end
168 end
169

```

3.4 Explicação do Programa

Para um melhor entendimento do programa uma breve explicação do mesmo se faz necessária.

Para a simulação, o usuário deve fornecer vetores com as informações dos tempos de chegada, tempo de computação, deadline e as informações sobre as zonas críticas.

As zonas críticas, associadas cada uma a um recurso exclusivo e um semáforo, devem ser fornecidas com tempos de chegada e de computação. O tempo de chegada é o tempo total de computação da tarefa que deve ter se passado para que ela entre numa zona crítica, esse tempo leva em conta outras zonas críticas pelas quais essa tarefa pode ter passado.

O simulador funciona em loop, que só acaba quando todas as tarefas terminam de ser computadas. A cada ciclo, o programa mede quanto tempo se passou desde o ciclo anterior, para simular o melhor possível um sistema em tempo real.

Ao início de cada loop o programa primeiro checa se alguma tarefa nova chegou, caso sim, ele a marca com uma tarefa ativa. Após isso, há uma checagem de qual é a tarefa ativa que possui maior prioridade (no caso o valor menor entre os dados pelo usuário) e essa é a tarefa que rodará nesse ciclo. A tarefa que roda no ciclo atual começa verificando se ela acabou de entrar em uma zona crítica. Caso tenha entrado, é preciso verificar se há algum ceiling $C(S^*)$ com prioridade maior ou igual à da tarefa atual entre os semáforos ativos, para bloquear e passar a prioridade da tarefa pra S^* , caso sim, ou permitir a entrada na zona crítica, caso não.

Após essa verificação, há a verificação de se há alguma zona crítica ativa ou não. Caso haja, o programa identifica qual é e desconta do seu contador de duração da zona crítica o tempo do ciclo, verificando se a zona crítica chegou ao fim, para poder liberar a tarefa de maior prioridade por ela bloqueada e reduzir a prioridade da tarefa atual de acordo com a lógica do PCP. Caso não haja zona crítica ativa, o tempo do ciclo é descontado do contador de duração da computação, verificando se a computação chegou ao fim, para desativar a tarefa.

Com essa forma de funcionamento, o simulador é capaz de simular o PCP ao longo de tantos loops quanto forem necessários para que todas as tarefas terminem de computar.

4 Exemplos de simulações

Nesta seção é possível observar algumas simulações feitas afim de exemplificar o PCP e assim verificar se a implementação do mesmo está correta.

4.1 Teste 1

4.1.1 Configurações

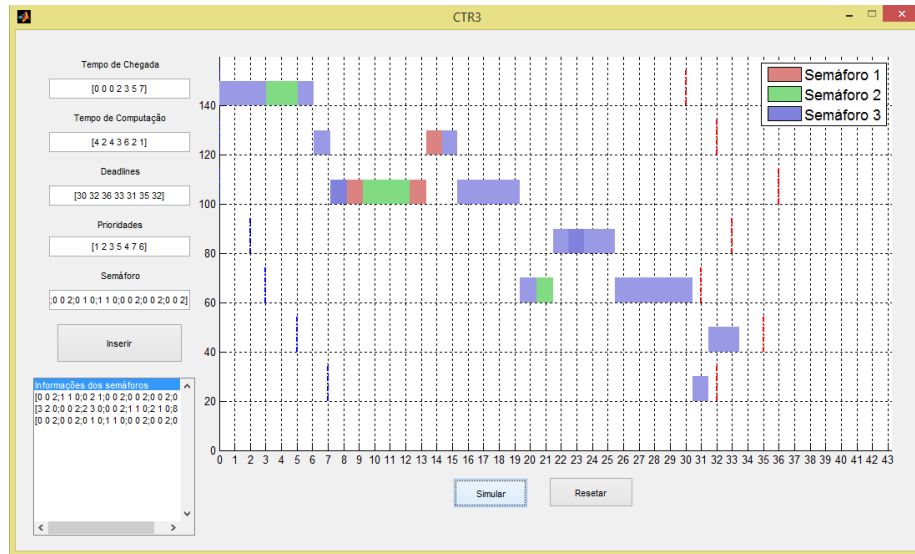
Tabela 1: *Tarefas.*

| Parâmetros | τ_1 | τ_2 | τ_3 | τ_4 | τ_5 | τ_6 | τ_7 |
|------------|----------|----------|----------|----------|----------|----------|----------|
| a_i | 0 | 0 | 0 | 2 | 3 | 5 | 7 |
| C_i | 4 | 2 | 4 | 3 | 6 | 2 | 1 |
| D_i | 30 | 32 | 36 | 33 | 30 | 40 | 31 |
| P_i | 1 | 2 | 3 | 5 | 4 | 7 | 6 |

Tabela 2: *Semáforo.*

| Tarefas | Sa_1 | Sa_C1 | Status 1 | Sa_2 | Sa_C2 | Status 2 | Sa_3 | Sa_C3 | Status 3 |
|----------|--------|---------|----------|--------|---------|----------|--------|---------|----------|
| τ_1 | 0 | 0 | 2 | 3 | 2 | 0 | 0 | 0 | 2 |
| τ_2 | 1 | 1 | 0 | 0 | 0 | 2 | 0 | 0 | 2 |
| τ_3 | 0 | 2 | 1 | 2 | 3 | 0 | 0 | 1 | 0 |
| τ_4 | 0 | 0 | 2 | 0 | 0 | 2 | 1 | 1 | 0 |
| τ_5 | 0 | 0 | 2 | 1 | 1 | 0 | 0 | 0 | 2 |
| τ_6 | 0 | 0 | 2 | 2 | 1 | 0 | 0 | 0 | 2 |
| τ_7 | 0 | 0 | 2 | 0 | 0 | 2 | 0 | 0 | 2 |

4.1.2 Resultado da Simulação



4.2 Teste 2

4.2.1 Configurações

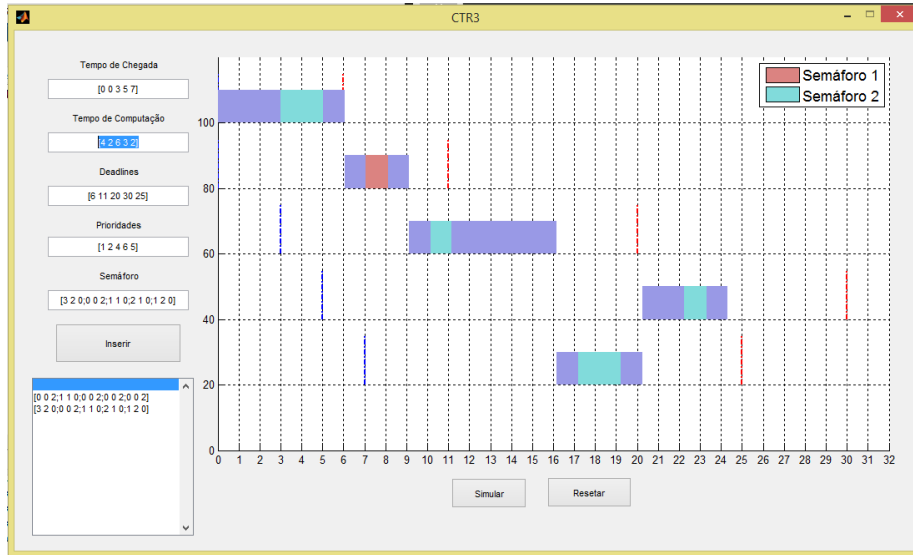
Tabela 3: *Tarefas.*

| Parâmetros | τ_1 | τ_2 | τ_3 | τ_4 | τ_5 |
|------------|----------|----------|----------|----------|----------|
| a_i | 0 | 0 | 3 | 5 | 7 |
| C_i | 4 | 2 | 6 | 3 | 2 |
| D_i | 6 | 11 | 20 | 30 | 25 |
| P_i | 1 | 2 | 4 | 6 | 5 |

Tabela 4: *Semáforo.*

| Tarefas | Sa_1 | Sa_C1 | Status 1 | Sa_2 | Sa_C2 | Status 2 |
|----------|--------|---------|----------|--------|---------|----------|
| τ_1 | 0 | 0 | 2 | 3 | 2 | 0 |
| τ_2 | 1 | 1 | 0 | 0 | 0 | 2 |
| τ_3 | 0 | 0 | 2 | 1 | 1 | 0 |
| τ_4 | 0 | 0 | 2 | 2 | 1 | 0 |
| τ_5 | 0 | 0 | 2 | 1 | 2 | 0 |

4.2.2 Resultado da Simulação



4.3 Teste 3

4.3.1 Configurações

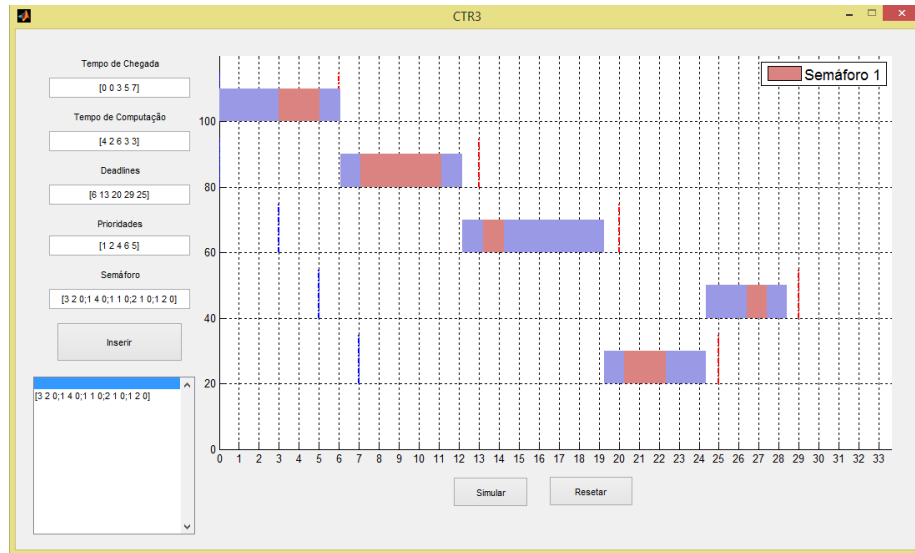
Tabela 5: *Tarefas*.

| Parâmetros | τ_1 | τ_2 | τ_3 | τ_4 | τ_5 |
|------------|----------|----------|----------|----------|----------|
| a_i | 0 | 0 | 3 | 5 | 7 |
| C_i | 4 | 2 | 6 | 3 | 3 |
| D_i | 6 | 13 | 20 | 29 | 25 |
| P_i | 1 | 2 | 4 | 6 | 5 |

Tabela 6: *Semáforo*.

| Tarefas | Sa_1 | Sa_C1 | Status 1 |
|----------|--------|---------|----------|
| τ_1 | 3 | 2 | 0 |
| τ_2 | 1 | 4 | 0 |
| τ_3 | 1 | 1 | 0 |
| τ_4 | 2 | 1 | 0 |
| τ_5 | 1 | 2 | 0 |

4.3.2 Resultado da Simulação



4.4 Teste 4

4.4.1 Configurações

Tabela 7: *Tarefas*.

| Parâmetros | τ_1 | τ_2 | τ_3 | τ_4 | τ_5 | τ_6 | τ_7 | τ_8 |
|------------|----------|----------|----------|----------|----------|----------|----------|----------|
| a_i | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C_i | 4 | 2 | 7 | 5 | 6 | 3 | 3 | 9 |
| D_i | 30 | 32 | 36 | 33 | 37 | 43 | 45 | 55 |
| P_i | 1 | 2 | 3 | 5 | 4 | 7 | 6 | 8 |

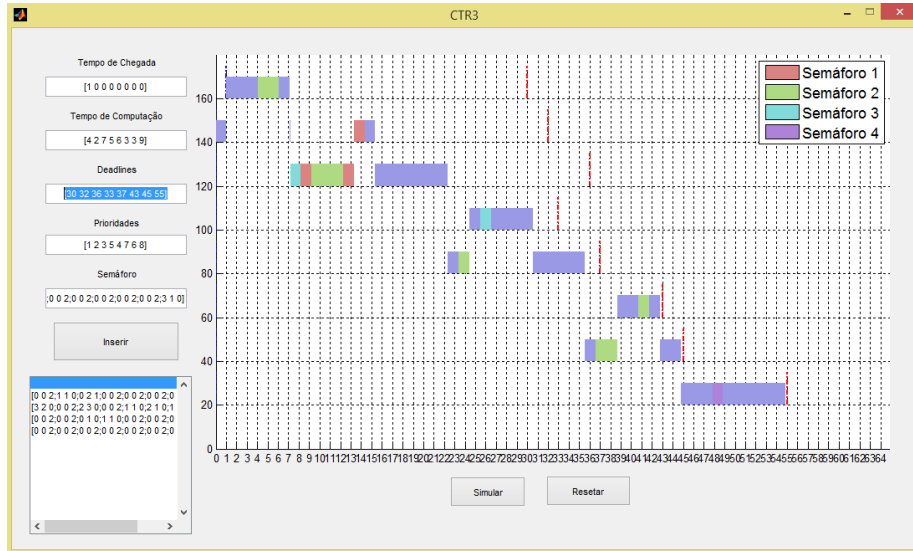
Tabela 8: *Semáforos 1*.

| Tarefas | Sa_1 | Sa_C1 | Status 1 | Sa_2 | Sa_C2 | Status 2 |
|----------|--------|---------|----------|--------|---------|----------|
| τ_1 | 0 | 0 | 2 | 3 | 2 | 0 |
| τ_2 | 1 | 1 | 0 | 0 | 0 | 2 |
| τ_3 | 0 | 2 | 1 | 2 | 3 | 0 |
| τ_4 | 0 | 0 | 2 | 0 | 0 | 2 |
| τ_5 | 0 | 0 | 2 | 1 | 1 | 0 |
| τ_6 | 0 | 0 | 2 | 2 | 1 | 0 |
| τ_7 | 0 | 0 | 2 | 1 | 2 | 0 |
| τ_8 | 0 | 0 | 2 | 0 | 0 | 2 |

Tabela 9: *Semáforos 2*.

| Tarefas | Sa_3 | Sa_C3 | Status 3 | Sa_4 | Sa_C4 | Status 4 |
|----------|--------|---------|----------|--------|---------|----------|
| τ_1 | 0 | 0 | 2 | 0 | 0 | 2 |
| τ_2 | 0 | 0 | 2 | 0 | 0 | 2 |
| τ_3 | 0 | 1 | 0 | 0 | 0 | 2 |
| τ_4 | 1 | 1 | 0 | 0 | 0 | 2 |
| τ_5 | 0 | 0 | 2 | 0 | 0 | 2 |
| τ_6 | 0 | 0 | 2 | 0 | 0 | 2 |
| τ_7 | 0 | 0 | 2 | 0 | 0 | 2 |
| τ_8 | 0 | 0 | 2 | 3 | 1 | 0 |

4.4.2 Resultado da Simulação



5 Anexo