Maria Inês Gomes Soares

Avaliação da Dependência / Independência dos Marcadores do Cromossoma Y

Faculdade de Ciências da Universidade do Porto
Agosto 2006

FC
Biblioteca
Faculdade de Ciências
Universidade do Porto

D000100844

Approved.
Maria Carvalho

Reg. 509266
Cota TESE Nº 166

Maria Inês Gomes Soares

Avaliação da Dependência / Independência dos Marcadores do Cromossoma Y

Faculdade de Ciências do Porto
MATEMÁTICA

Departamento de Matemática Pura
Faculdade de Ciências da Universidade do Porto
Agosto de 2006

Maria Inês Gomes Soares

Avaliação da Dependência / Independência dos Marcadores do Cromossoma Y

Tese submetida à Faculdade de Ciências da Universidade do Porto
para obtenção do grau de Mestre em Engenharia Matemática
orientada pelo Professor Doutor Pedro Silva
e co-orientada pelo Professor Doutor António Amorim e pela Doutora Leonor Gusmão

Departamento de Matemática Pura
Faculdade de Ciências da Universidade do Porto
Agosto de 2006

Agradecimentos

Este estudo só foi possível porque pude contar, desde o princípio, com a cooperação, apoio, solidariedade, estímulo e incentivo de algumas pessoas. A elas os meus profundos agradecimetos.

Ao **Professor Doutor Pedro Silva**, pelo forma como orientou a execução deste trabalho e por ter estado sempre presente, numa atitude de suporte incondicional, reveladora de uma honestidade intelectual, conhecimento e generosidade. Atitude de um mestre que pretende o crescimento científico e humano do seu pupilo.

Ao **Professor Doutor António Amorim** e à **Doutora Leonor Gusmão**, meus co-orientadores, pela presença, disponibilidade e críticas sempre oportunas durante a realização deste trabalho.

A **todos os Professores do Mestrado**, pela contribuição para a minha formação académica.

Aos **meus Colegas de Mestrado**, pelas trocas de experiências, que me enriqueceram como profissional e como pessoa.

À **FCUP**, por me ter dado a oportunidade de crescer academicamente.

Aos **meus Amigos**, pelo apoio e incentivo.

Aos **meus Pais e Irmãos**, pelo carinho demonstrado, apoio, incentivo e por me ensinarem que "querer é poder".

Ao **Gonçalo Oliveira**, pelo seu amor, carinho, compreensão, apoio e incentivo.

Resumo

O presente trabalho foi elaborado com o intuito de responder a diversas questões levantadas na área da genética, tendo em consideração as mutações ocorridas no *cromossoma Y*.

Num contexto genético, assume-se a existência de associação entre alelos de diferentes *loci*, e uma vez que o *cromossoma Y* não apresenta recombinação, pensa-se que as mutações no mesmo são as responsáveis pela quebra das associações referidas. Assim sendo, e particularizando o caso dos marcadores *Y-STRs* (*STR - Short Tandem Repeat*), é de esperar que ao analisar *haplótipos* do *cromossoma Y*, alguns destes marcadores já não apresentem qualquer tipo de associação.

No sentido de se poder testar esta hipótese, foi estudada uma amostra de *658 indivíduos*, analisados para *15 Y-STRs*. Com esta finalidade, foram criados diferentes programas em *Python*, nos quais, assumindo sempre independência entre os diferentes marcadores, se usou o *teste do Qui-Quadrado*, com o intuito de testar a verosimilhança da *Independência/Dependência* entre os diversos *Y-STRs*.

Os resultados do estudo reflectem um número elevado de casos de dependência entre os diferentes marcadores, quando se assume contiguidade entre os diversos valores, constatando-se um aumento da dependência quando se agrupam três marcadores em vez de dois. É notável, também, que a não exigência da contiguidade de valores não provocou grandes alterações nos resultados. Analisando os resultados obtidos da junção dos dois estudos efectuados para dois marcadores e três marcadores, evidencia-se também um número reduzido de casos que manifestam dependência entre três marcadores, mas independência entre os mesmos, quando considerados dois a dois, sob a condição ou não da contiguidade de valores. No que respeita à independência entre marcadores, considerados

em dimensão superior ou igual a quatro, não é possivel observar independência entre os mesmos exigindo também uma independência entre os subgrupos de dimensões inferiores. Verificou-se também um número bastante reduzido de marcadores independentes dois a dois, quando é imposta a soma entre os valores nos mesmos, no entanto este número diminui quando, sob as mesmas condições, se considerou três marcadores em vez de dois. Para terminar, testou-se a independência de um marcador contra a soma dos restantes, e o resultado obtido evidenciou sempre dependência entre os mesmos.

Os resultados do estudo evidenciam um considerável número de casos de dependência, independentemente de qualquer que seja a condição imposta ao estudo.

Abstract

The current work was developed with the aim of answering several questions raised in genetics, taking into consideration the mutations occurring in the *chromosome Y*.

In a genetic context, we assume the existence of association between alleles of different *loci*. Since the *chromosome Y* does not present recombination, it is assumed that its mutations cause the break of the mentioned associations. Being so, and particularly in the case of the *Y-STRs markers (Short Tandem Repeats)*, it is likely that, whenever analysing haplotypes of the *chromosome Y*, some of these markers no longer do exhibit any type of association.

With the purpose of testing this hypothesis we studied a sample of *658 individuals*, analysed for *15 Y-STRs*. With this aim, several *Python* programs were developed, upon which, under the assumption of independence between markers, we used the χ^2 - *test* to evaluate the likelihood of *Dependency/Independency* between the various *Y-STRs*.

The study's results reflect a large number of dependency cases between different markers, when we assume contiguity between different values, witnessing an increase of dependency when going from two to three markers. It should be remarked that the contiguity argument does not weight too much in the final output. Analysing the results obtained by combination of the two and three markers cases, we conclude that only a reduced number of cases present level three dependency whenever they are level two independent for all possible subcases, taking into consideration or not the contiguity condition. With respect to marker independency for dimension ≥ 4, no case of independency arises from lower dimension independency subcases. We observed also a low number of level two independency cases with respect to sum, getting even smaller when we consider three markers. Finally, we tested the independency of a marker against the sum of the others,

Faculdade de Ciências do Porto
MATEMÁTICA

revealing dependency in all cases.

The results obtained suggest dependency in most cases, for all conditions considered.

Résumé

Le présent travail fut développé pour répondre à plusieurs questions dans le domaine de la génétique, considérant des mutations observées dans le *chromosome Y*.

Dans un contexte génétique, on assume l'existence d'association entre allèles de différents *loci*. Comme le *chromosome Y* ne présents point de récombination, on croit que ses mutation entraînent les associations référées. Dans le cas particulier des marqueurs *Y-STRs (Short Tandem Repeats)*, on s'attend à qu'en analysant des haplotypes du *chromosome Y*, quelques de ces marqueurs ne présentent plus aucun type d'association

Pour pouvoir tester cette hypothèse, on a étudié un fichier comprenant *658 individus*, analysés pour *15 Y-STRs*. Avec ce but, on a écrit plusieurs programmes en *Python*, dans lesquels, avec l'assomption d'indépendance entre les différents marqueurs, on a utilisé le *test* χ^2 pour mesurer la vraisemblance *d'Indépendance/Dépendance* entre les différents *Y-STRs*.

Les résultats de l'étude réflectent un grand nombre de cas de dépendance entre les différents marqueurs, quand on assume contiguïté entre les différents valeurs, en constatant le renforcement de la dépendance en passant au niveau trois. On doit aussi remarquer que l'exigence de contiguïté ne provoque pas de grandes altérations dans les résultats. Considérant l'ensemble des résultats niveau deux et niveau trois, il devient clair qu'il y a un nombre très petit de cas qui manifestent dépendance au niveau trois et indépendance au niveau deux dans ses différentes possibilités, soit qu'on considère la condition de contiguïté ou pas. Dans ce qui concerne l'indépendance des marqueurs, en dimension ≥ 4, on n'a pas constaté des cas d'indépendance entraînant indépendance aux niveaux inférieurs. On a remarqué aussi un nombre très petit de marqueurs indépendants au niveau deux en considérant leur somme. Ce nombre est encore plus petit quand on considère la somme de trois marqueurs. Finalement, on a testé l'indépendance d'un marqueur contre la somme

des autres, obtenant toujours des cas de dépendance.

Les résultats obtenus témoignent un très grand nombre de cas de dépendance, pour la généralité des conditions considérées.

Conteúdo

1 Introdução **15**

2 Noções básicas **18**

2.1 Genética 18

2.2 Teste do *Qui-Quadrado* 21

3 Programas **26**

3.1 Dados.py 26

3.2 2colunas_Contiguidade.py 34

3.3 3colunas_Contiguidade.py 49

3.4 2colunas+3colunas_Contiguidade.py 62

3.5 2colunas_Não Contiguidade.py 63

3.6 3colunas_Não Contiguidade.py 71

3.7 2colunas+3colunas_Não Contiguidade.py 78

3.8 Ordenar por Probabilidades.py 80

3.9 Soma de n colunas.py 94

3.10 Soma das Diferenças nos Extremos.py 112

3.11 Independência de 2 colunas e Sucessivamente.py 125

3.12 Um Contra Todos.py 149

4 Conclusão **158**

5 Relevância do Resultado para o IPATIMUP **161**

A Tabela do Qui-Quadrado **164**

B Tabela YPortugal_Total2.csv **168**

C Esquema - Dados.py **188**

D Esquemas - Independência de 2 Colunas e Sucessivamente.py **189**

Bibliografia **193**

Capítulo 1

Introdução

A Genética é um ramo da ciência que se dedica ao estudo da hereditariedade, isto é, da transmissão de características de uma geração para as seguintes e das variações hereditárias. É de notar que, a informação genética não é transmitida fielmente de geração em geração. O genoma de um indivíduo pode sofrer alterações designadas *mutações*. Nos sistemas genéticos sem recombinação este é o único processo que gera diversidade, como é o caso do *cromossoma Y*. As mutações podem contribuir para a quebra de associação, em particular no caso dos marcadores do tipo *Y-STR*.

Em termos práticos, a comprovação de ausência de associação entre alguns dos *Y-STRs* mais utilizados na prática forense é de extrema importância, uma vez que contribui para uma valoração mais objectiva da evidência genética em casos de identificação ou na determinação de relações de parentesco biológico.

O objectivo fulcral deste trabalho consiste na averiguação da hipótese de Independência entre os diferentes marcadores do *cromossoma Y*, do tipo anteriormente referido, principalmente para aqueles que apresentam uma maior taxa de mutação.

Neste sentido será estudada uma amostra de *658 indivíduos*, analisados para *15 Y-STRs*, fornecida pelo *IPATIMUP*. Para tal, serão elaborados diferentes programas em *Python*, com vista a responder às mais variadas questões que surgem neste contexto. Na execução dos mesmos, pretende-se usar o *teste do Qui-Quadrado*, que consiste na comparação do *valor do Qui-Quadrado obtido* com o *tabelado* (note-se que o valor tabelado, aconselhado

para a comparação, diz respeito ao nivel de significância 0.05 - corresponde a considerar razoável correr o risco de rejeitar uma *hipótese nula* verdadeira, com probabilidade 0.05, ou seja, uma em cada 20 vezes, em termos médios). Salienta-se também que na aplicação do referido teste se assumirá, à priori, independência entre os diferentes marcadores - *Hipótese nula.*

Os programas elaborados podem ser utilizados com qualquer ficheiro de dados típico neste contexto. Como exemplo, será estudada a amostra fornecida pelo *IPATIMUP*, que consiste num ficheiro *Excel* com *658 linhas*, que representam os *658 indivíduos*, e *17 colunas*, das quais as duas primeiras representam a região e o número de identificação de cada indivíduo, e as restantes *15 colunas* os *15 Y-STRs.*

Tal como foi referido anteriormente, no presente trabalho serão desenvolvidos diversos programas em *Python*, numa totalidade de *12 programas*, de entre os quais um será criado com vista a garantir um acesso eficiente aos dados do ficheiro necessários à execução dos restantes programas.

Assim, como primeira sugestão surge a elaboração de programas para testar a verosimilhança da Independência/Dependência de dois marcadores/colunas e três marcadores/colunas, nos quais se pretendem agrupar os valores *contíguos*, com frequências esperadas menores que *5*, visto estes valores poderem comprometer a fiabilidade dos resultados a obter e, para testar a influência desta condição, serão elaborados programas nas mesmas condições, mas sem supor a contiguidade de valores. Posteriormente, no caso em que se supõe contiguidade e no caso em que não se supõe essa mesma condição, pretende-se aplicar o *teste do Qui-Quadrado* para testar a dependência de três colunas sem que as mesmas duas a duas o sejam. Ainda dentro do mesmo contexto será elaborado um outro programa para testar a hipótese de Dependência/Independência, para duas colunas nas mesmas condições, apenas para o caso em que se supõe a contiguidade de valores, mas no qual se ordenam os dados por ordem decrescente de intervalos de probabillidade, depois graus de liberdade, e por fim valor do *Qui-Quadrado* determinado.

Surge ainda a sugestão para a elaboração de mais dois programas, com vista a testar a Dependência/Independência de *n colunas*, impondo a soma dos valores nas mesmas. Num deles pretende-se impor, ainda, a soma nos extremos, isto é, a soma mínima e a soma máxima possível nas respectivas colunas.

Será, ainda, elaborado um outro programa com o intuito de testar a Independência/Dependência das colunas duas a duas e, posteriormente, apenas para estas, testar a Independência/Dependência três a três, e assim sucessivamente. O objectivo na criação deste programa consiste em testar a verosimilhança da Independência de *n colunas*, tendo em conta que estas o seriam quando consideradas em subgrupos de *n-1 colunas*.

Para terminar, pretende-se, ainda, desenvolver um outro programa com vista a testar a hipótese de Dependência/Independência de uma coluna em relação à soma das restantes.

Após a execução dos programas citados, e uma posterior análise dos resultados, pretende-se obter uma resposta à questão focal do trabalho, isto é, pretende-se avaliar o grau de dependência entre as distribuições alélicas de cada locus do tipo *Y-STR*.

Como já era de esperar, pelas hipóteses previamente avançadas pelos investigadores do *IPATIMUP*, os resultados do estudo evidenciam um grau de dependência bastante elevado entre os marcadores do tipo *Y-STR*, independentemente das condições impostas ao estudo.

Capítulo 2

Noções básicas

2.1 Genética

O trabalho de cientistas como *Mendel*, *Morgan*, *De Vries*, *Watson* e *Crick*, e de muitos outros investigadores, permitiu saber que as informações hereditárias são transmitidas ao longo das gerações, segundo determinados padrões, e que essa informação tem um suporte físico - *o material genético.*

O *material genético* encontra-se em macromoléculas poliméricas, designadas *DNA* (*ácido desoxirribonucleico*).

Em 1953 *James Watson* e *Francis Crick*, associando dados químicos obtidos por *Chargaff* e os dados obtidos por *Wilkins* por meio de difracção de raio X, propuseram um modelo de estrutura do *DNA*, que lhes valeu o prémio Nobel em 1962. A molécula de *DNA* é formada por duas cadeias polinucleotídicas enroladas uma em volta da outra, no sentido dos ponteiros do relógio, formando uma dupla hélice. Cada cadeia polinucleotídica é formada por nucleótidos, os quais são compostos por uma pentose (desoxirribose), um grupo fosfato e uma base azotada (esta pode ser: *adenina -A*, *guanina - G*, *citosina - C* e *timina- T*). *Watson* e *Crick* afirmaram também que as bases de cada cadeia polinucleotídica estavam orientadas para o interior de cada cadeia dupla, ligando-se às bases da outra cadeia por ligações de hidrogénio. Os emparelhamentos ocorrem apenas entre a *adenina - A* e a *timina- T* e a *guanina - G* e a *citosina - C*, o que determina uma rigorosa complementariedade

entre as bases. A grande diversidade das sequências de *DNA* determina a variabilidade desta molécula nos diversos seres vivos.

Cada segmento de *DNA* que contém informação para sintetizar uma determinada proteína designa-se *gene*.

Nos organismos eucariontes, a informação genética encontra-se distribuida por várias moléculas de *DNA*, as quais estão associadas a proteínas designadas *histonas*. Cada porção do *DNA* associado às *histonas* constitui um filamento de *cromatina*. Estes filamentos encontram-se, na maior parte do tempo, dispersos no núcleo da célula. No entanto, quando a célula está em divisão, estes filamentos sofrem um processo de condensação, originando filamentos curtos e espessos, os quais se passam a designar por *cromossomas*. Assim, pode-se considerar que os cromossomas são estruturas presentes no núcleo das células, que contêm os *genes*. O local do cromossoma onde existe um determinado *gene* que codifica uma determinada característica designa-se *locus* (no plural *loci*). Designam-se por *genes alelos*, aqueles que têm a informação para a mesma característica no mesmo *locus*.

Cada espécie caracteriza-se pelo número de cromossomas por célula e pela sua morfologia (forma e tamanho) - *Cariótipo*.

O *cariótipo* humano é constituído por *46 cromossomas* (*23 pares*). Destes *46 cromossomas*, *44* são *autossomas* (*22 pares*) e *2* são *heterossomas* ou *cromossomas sexuais* (*1 par*). Estes dois *cromossomas* são os responsáveis pela definição do sexo de um indivíduo. Os indivíduos do sexo masculino possuem um *cromossoma X* e um *cromossoma Y*, enquanto os indivíduos do sexo feminino possuem dois cromossomas *X*. O *cromossoma X* apresenta dimensões muito superiores às do *cromossoma Y*.

A informação genética dos indivíduos não é, contudo, estática no tempo, podendo ser alterada por uma série de factores. Qualquer alteração na informação genética de um indivíduo designa-se por *mutação*. Sob o ponto de vista molecular a *mutação* traduz qualquer alteração na sequência de pares de bases no *DNA*.

As mutações podem ocorrer espontanemente ou serem induzidas por diversos agentes mutagénicos, que podem ser de natureza física (como as radiações solares, as radiações cósmicas e as radiações resultantes da desintegração atómica) ou de natureza química (como o gás mostarda).

As mutações espontâneas mais comuns provêm de erros ocorridos durante a replicação, despurinações, desaminações e oxidações de bases do *DNA*. Estas ocorrem a uma frequência muito baixa e, portanto, para estudá-las laboratorialmente recorre-se a agentes mutagénicos (físicos e químicos) para aumentar consideravelmente a taxa de mutação e obter organismos mutantes em quantidade suficiente para análise genética.

O efeito de uma mutação numa célula pode ser tão pequeno que não seja possível evidenciar-se facilmente, mas também pode ser tão significativo que conduza à morte da célula ou do organismo. Embora as nossas células tenham capacidade para reparar os dados produzidos por uma mutação, por vezes o equilíbrio entre mutagénese e reparação rompe-se.

Imaginando uma sequência inicial, as mutações vão criando sequências progressivamente divergentes, a menos que haja reversão (isto é, por exemplo que a sequência TAAAAAA sofresse uma mutação que restaurasse o estado inicial AAAAAAA).

No caso em estudo, tem-se em consideração as mutações ocorridas no *cromossoma Y*. Assume-se que existirá uma associação entre alelos de diferentes *loci*, e uma vez que o *cromossoma Y* não apresenta recombinação, pensa-se que as mutações no mesmo serão as responsáveis pela quebra das associações referidas. Particulariza-se o caso dos *marcadores* do tipo *STR - Short Tandem Repeat* que se caracterizam pela presença de motivos repetitivos, cujo número de cópias varia entre indivíduos. Por exemplo, para um *STR* representado por $(GATA)_n$ os indivíduos de uma população poderão possuir diferente número de cópias do motivo *GATA* (por exemplo, este motivo de bases poderá estar repetido 8, 9 ou 10 vezes). Assim sendo, é de esperar que ao analisar *haplótipos* do *cromossoma Y* defenidos por *STRs*, alguns destes marcadores, especialmente aqueles que apresentarem maiores taxas de mutação, já não apresentem qualquer tipo de associação alélica.

Em termos práticos, a comprovação de ausência de associação entre alguns dos *Y-STRs* mais utilizados na prática forense é de extrema importância, pois permitirá uma estimativa mais fiável das frequências haplotipicas (a partir do produto das frequências alélicas para *loci* não associados - este facto encontra-se explicado com mais pormenor na secção seguinte), contribuindo assim para uma valoração mais objectiva da evidência genética em casos de identificação ou na determinação de relações de parentesco biológico.

No sentido de se poder testar esta hipótese, é estudada uma amostra de *658 indivíduos*, analisados para *15 Y-STRs*.

A *Tabela 1* é um extracto da tabela fornecida pelo *IPATIMUP* armazenada no ficheiro *YPortugal_Total2.csv* (*Apêndice B*), o qual é usado na execução dos diferentes programas elaborados para estudar a hipótese anteriormente referida.

Tabela 1: Extracto do ficheiro *YPortugal_Total2.csv*

Note-se que cada linha representa um *indivíduo* e cada coluna um *marcador* do tipo *Y-STR*.

Reg.	IDNr.	Y19	Y389I	Y389II	Y390	Y391	Y392	Y393	Y437	Y438	Y439
B	2445	13	13	17	24	10	11	13	14	10	13
VC	3700	16	12	17	22	9	12	13	16	10	13
VC	3495	13	13	18	24	10	11	12	14	10	12
VC	3692	13	14	19	24	10	11	12	14	10	12
B	528	13	12	17	24	10	11	13	14	10	10
BN	756	13	13	17	24	10	11	12	14	10	12
P	1174	13	13	18	23	10	11	13	14	10	12
P	1238	13	12	17	24	10	11	13	14	10	10
P	1290	13	13	17	24	10	11	12	14	10	12

2.2 Teste do *Qui-Quadrado*

Existem vários testes que permitem testar a independência entre variáveis categóricas, tais como: Testes do Qui-Quadrado, Teste de Kolmogorov-Smirnov, Teste de aleatoriedade de Fisher, Teste de discordância de McNemar, Teste dos Sinais, Teste de Wilcoxon, Teste de Wilcoxon-Mann-Whitney, Teste de Siegel-Tukey, Teste de Kruskal-Wallis, Teste de Friedman, de entre outros.

Todavia, no caso de uma variável (*Discreta*) tomando um número finito de valores, os testes baseados na *distribuição do Qui-Quadrado*, denominados *Testes do Qui-Quadrado*, são os mais usados, pelo que, no presente trabalho, se usa o *Teste do Qui-Quadrado* com o intuito de averiguar a hipótese de Dependência/Independência entre os marcadores do *Cromossoma Y*, do tipo *STR*.

Em seguida encontra-se uma breve explicação sobre o referido método.

O Teste do *Qui-Quadrado* - χ^2 - é muito eficiente para avaliar a associação entre variáveis qualitativas.

O princípio básico deste método não paramétrico reside na comparação das divergências entre as frequências observadas e as esperadas. De uma maneira geral, pode-se dizer que dois grupos se comportam de uma forma semelhante se as diferenças entre frequências observadas e esperadas, em cada categoria, forem muito pequenas, isto é, próximas de zero.

O valor do *Qui-Quadrado* é calculado da seguinte forma:

$$Q = \sum \frac{(obs - esp)^2}{esp}$$

onde *obs* representa as frequências observadas e *esp* as esperadas.

Note-se que a diferença $obs - esp$ será muito pequena quando as frequências observadas forem muito próximas das esperadas, e será muito grande quando a discrepância entre as mesmas for muito grande. Neste último caso, o valor do *Qui-Quadrado* calculado, Q, tomará valores muito altos.

É de notar, ainda, que na aplicação do teste do *Qui-Quadrado* se consideram sempre duas hipóteses:

$$H_0 : \textit{Caso de Independência}$$

$$H_1 : \textit{Caso de Dependência}$$

Assim, a aplicação do referido teste consiste na comparação do valor do *Qui-Quadrado* calculado, Q, com o tabelado, χ^2, tendo em conta os graus de liberdade, e o nível de significância. Se o primeiro valor for maior ou igual que o segundo, isto é, $Q \geq \chi^2$, então

H_0 é rejeitada, ou seja, está-se sob um caso de *Dependência* e as frequências esperadas não terão de ser semelhantes às frequências observadas, caso contrário, aceita-se H_0 e está-se sob um caso de *Independência* e, neste caso, as frequências esperadas deverão ser semelhantes às frequências observadas.

Em seguida, explica-se de forma mais pormenorizada o cálculo do valor do *Qui-Quadrado*, Q, assim como dos graus de liberdade e do nível de significância, tendo em conta o ficheiro excel fornecido pelo *IPATIMUP*, $YPortugal_Total2.csv$, no qual cada linha representa um indivíduo e cada coluna representa um marcador do cromossoma Y, como já foi referido na secção anterior.

As *frequências observadas* são obtidas directamente dos dados expressos no ficheiro mencionado, por contagem das posições nas condições pretendidas, enquanto que as frequências esperadas são calculadas a partir destas.

Note-se, primeiramente, que se trabalha sempre sob a condição da hipótese nula, isto é, H_0, ser verdadeira, ou seja, que se está sob um caso de Independência, pelo que faz sentido considerar a probabilidade da intersecção de casos, como sendo o produto das probabilidades dos diferentes casos. Assim, se A e B forem dois casos a considerar, tem-se:

$$P(A \cap B) = P(A) \times P(B)$$

Assim sendo, na determinação das frequências esperadas basta multiplicar as probabiliades dos casos que estão a ser considerados, pelo número total de linhas do ficheiro já referido.

Convém referir, também, que a probabilidade de cada caso considerado consiste na razão entre o número de vezes que esse caso é verificado, nas condições pretendidas, pelo número total de linhas do ficheiro.

No que diz respeito aos graus de liberdade, no caso do espaço de possibilidades admitir uma decomposição como produto directo, estes são determinados multiplicando as dimensões dos casos que estão a ser considerados, subtraindo uma unidade a cada factor. Caso contrário, subtrai-se uma unidade à dimensão.

Convém, ainda, referir que não existem regras de decisão cem por cento seguras, no que respeita à aceitação ou rejeição da *hipótese nula* - H_0. Esta pode ser rejeitada, sendo verdadeira - neste caso diz-se que se tem um erro de *1ª espécie*, - ou pode ser aceite, sendo falsa - neste caso, dir-se-ía um erro de *2ª espécie*. Não é possível controlar as probabilidades de ambos os tipos de erro, por isso, fixa-se a probabilidade p de cometer um erro de *1ª espécie*, a que se chama *nível de significância do teste.*

Tal como já foi referido, depois de determinado o valor do *Qui-Quadrado*, Q, compara-se o valor obtido com o tabelado, χ^2, tendo em conta os graus de liberdade calculados e o nível de significância. No caso de $Q \geq \chi^2$, então H_0 é improvável e os casos deverão ser dependentes. No caso de $Q \prec \chi^2$, então H_0 é provável e os casos deverão ser independentes. De forma mais rigorosa, se $\chi^2_{gl,p}$ for o valor tabelado, correspondente a gl graus de liberdade, p representa a probabilidade de obter $Q \geq \chi^2_{gl,p}$.

Note-se que o ficheiro considerado para a realização do teste do *Qui-Quadrado*, *Qui_Quadrado.csv* (*Apêndice A*) tem seis colunas, onde a primeira diz respeito aos graus de liberdade, e as restantes dizem respeito aos níveis de significância, isto é, às probabilidades, respectivamente, 0.1, 0.05, 0.025, 0.01 e 0.005. Assim, a cada posição da tabela está associado um determinado grau de liberdade, gl, e uma determinada probabilidade, p, pelo que esta posição representa o valor do *Qui-Quadrado* tabelado, com gl graus de liberdade e nível de significância p. É comum usar o valor $p = 0.05$ para distinguir dependência e independência. Nestas condições, a hipótese nula, H_0 seria de rejeitar, se $Q \geq \chi^2_{gl,0.05}$.

Convém referir que no caso de surgirem frequências esperadas com valores menores que 5 estas poderão criar algumas distorções na obtenção dos resultados, comprometendo a sua fiabilidade. Consequentemente, na eventualidade das mesmas surgirem, proceder-se-á à sua aglomeração. Esta preocupação será visivel em vários programas, e é deduzida em conformidade com o *Teorema* que em seguida se enuncia.

Teorema 2.2.1 *Quando a hipótese nula - H_0 - é verdadeira, a estatística do teste, dada pelo valor Q definido anteriormente, tem assintoticamente distribuição do Qui-Quadrado, em que os graus de liberdade se determinam da forma acima referida.*

O *Teorema* enunciado é deduzido supondo n *(número de dados a considerar)* $\rightarrow \infty$. Assim, para que a aproximação seja válida no *caso finito*, as frequências esperadas nunca

devem ser inferiores a 5. Se, como muitas vezes acontece, as frequências esperadas de algumas classes extremas forem inferiores a este número, essas classes devem agrupar-se com as adjacentes, sendo o número de graus de liberdade reduzido em conformidade.

Capítulo 3

Programas

3.1 Dados.py

Este programa foi elaborado com o objectivo de garantir um acesso eficiente aos dados necessários à execução dos restantes programas. Com esta finalidade, começa-se por introduzir o nome do ficheiro com que se pretende trabalhar, *ficheiro*, seguindo-se a respectiva abertura do mesmo, *f*, com vista a possibilitar a execução dos passos seguintes. É de salientar que, neste trabalho, o ficheiro em causa é $YPortugal_Total2.csv$.

```
# Chamar o ficheiro de texto YPortugal_total2.csv, que contém os dados necessários.

ficheiro = raw_input("Escreva o nome do ficheiro:\n)
f = open( ficheiro )
```

Posteriormente, é criado um ciclo, no qual se leêm as linhas do ficheiro *f*, e as quais se armazenam num dicionário denominado por *s*. Em seguida, é criado um outro dicionário, *t*, no qual os espaços brancos que aparecem no dicionário *s* são convertidos em vírgulas. Assim, cada elemento *t[i][j]* do dicionário *t*, representa o valor do ficheiro que se encontra na linha *i* e coluna *j*. Note-se ainda que cada *t[i]* representa a linha *i* do ficheiro, na qual os diferentes elementos estão separados por vírgulas.

Por exemplo:

t[1]=['B','2445','13','13','17','24','10','11','13','14','10','13','16','9','12','22','28']

t[1][2]=13

representam a *2ª linha* do ficheiro mencionado anteriormente (note-se que em *Python* a contagem começa em 0, pelo que a *linha 1* representa a *2ª linha* do ficheiro) e o *3º elemento* dessa mesma linha, respectivamente.

Ainda no mesmo ciclo é determinado o número total de linhas do ficheiro. À medida que as linhas são lidas, *t[i]*, é verificado se estas são diferentes da lista vazia, isto é, *[]*. Em caso afirmativo vai-se adicionado *1* a *i* e a *nl*, variáveis que inicilmente tomam o valor *0*, e correspondem, respectivamente, à linha que se está a considerar, e ao número total de linhas do ficheiro, quando *t[i]* começar a ser igual a *[]*.

O número total de colunas do ficheiro, *nc*, corresponde ao comprimento de cada *t[i]*. (Note-se que, para cada linha *i*, *t[i]* tem sempre o mesmo comprimento, pelo que basta determinar *nc* para um *i* qualquer, por exemplo *0*.)

```
# O dicionário s representa as linhas do ficheiro.
# O dicionário t representa as mesmas linhas,mas sem os espaços em branco, estes são substituidos por vírgulas.
# t[i][j] representa o elemento que se encontra na linha i e coluna j.
# nl dá-nos o número de linhas do ficheiro.
# nc dá-nos o número de colunas do ficheiro.

s = {}
t = {}
i = 0
nl = 0
while (0 == 0):
    s[i] = f.readline()
    t[i] = s[i].split( )
```

```
    if (t[i] != []):
        nl = nl + 1
        i = i + 1
    else:
        break
nc = len (t[0])
#print "nl =", nl
#print "nc =", nc
#for i in range( nl ):
    #for j in range( nc ):
        #print "t(", i, ",", j, ") =", t[i][j]
```

Seguidamente, de entre o número total de linhas, *nl*, e do número total de colunas, *nc*, do ficheiro determinam-se aquelas linhas e colunas que não convém considerar na execução dos diferentes programas.

Começa-se por considerar uma lista formada pelos algarismos de *0* a *9*, a qual se denomina *alfabeto*. Como foi visto anteriormente, cada linha *i* é representada por uma lista *t[i]*. Assim, as linhas não consideradas dizem respeito àquelas que têm, em cada coluna, digitos diferentes dos que constam na lista *alfabeto* ou digitos iguais a ". Como as colunas são semelhantes, no que diz respeito aos digitos que nelas aparecem, é conveniente considerar apenas a última coluna, na determinação do número de linhas não consideradas. Assim sendo, tal como já foi referido, vê-se, para cada linha, se aparece algum digito diferente dos que constam na lista alfabeto, ou algum digito igual a ", sempre na última coluna. No caso de uma destas condições se verificar, soma-se *1* à variável *G*, que inicialmente toma o valor *0*. No final, a variável *G* representará o número de linhas não consideradas do ficheiro, e denotar-se-á *nl_nao*.

Posteriormente, segue-se um processo identico na determinação do número de colunas a não considerar. À semelhança do que acontece com as colunas, na determinação de *nl_nao*, com as linhas passa-se o mesmo, na determinação do número de colunas a não considerar. Assim sendo, basta considerar apenas uma linha para a realização dos cálculos.

Para este efeito, considera-se então a primeira linha a seguir à última não considerada. Assim, para cada coluna dessa mesma linha, começa-se por ver se o número de dígitos presentes na mesma é maior que *2*. No caso de tal condição se verificar, então essa coluna não será considerada. Caso contrário, isto é, se o número de digitos presentes na coluna for menor ou igual a *2*, então impõe-se outra condição, verificar se cada um dos digitos presentes na coluna se encontra na lista *alfabeto*. Em caso negativo, esta coluna não será considerada. Note-se ainda que cada vez que surge uma coluna a não considerar, soma-se *1* à variável *GG*, que inicialmente toma o valor *0*. Da mesma forma que na determinação de *nl_nao*, no final a variável *GG* representará o número de colunas a não considerar, e denotar-se-á por *nc_nao*.

```
# alfabeto é a lista com os algarismos de 0 a 9.
# nl_nao dá-nos o número de linhas do ficheiro não consideradas.
# nc_nao dá-nos o número de colunas do ficheiro não consideradas.
# Note-se que para determinar o nc_nao, basta trabalhar apenas com uma linha. Trabalhou-
se com a 1ªlinha que é considerada.
alfabeto = ["0","1","2","3","4","5","6","7","8","9"]
G = 0
for i in range (nl):
    for ii in range (len(t[i][nc-1])):
        if ((t[i][nc-1][ii] == "") or (t[i][nc-1][ii] not in alfabeto)):
            G = G + 1
            break
nl_nao = G
#print "nl_nao =", G
GG = 0
for j in range(nc):
    if (len(t[nl_nao][j]) > 2):
        GG = GG + 1
    else:
```

```
        for ii in range(len(t[nl_nao][j])):
            if (t[nl_nao][j][ii] not in alfabeto):
                GG = GG + 1
                break
nc_nao = GG
#print "nc_nao =", nc_nao
```

Salienta-se que nos próximos passos deste programa, assim como nos programas seguintes, apenas se trabalhará com as linhas posteriores a *nl_nao*, inclusive, até *nl*, e com as colunas posteriores a *nc_nao*, inclusive, até *nc*.

Em seguida são calculados o menor e o maior valor que aparece na tabela. Com essa finalidade, começa-se por considerar como valor mínimo, *min*, e como valor máximo, *max*, o primeiro valor que aparece na tabela, isto é, o valor que se encontra na primeira linha e primeira coluna, a serem consideradas. Assim, *min* e *max* são inicialmente iguais, e representados por *t[nl_nao][nc_nao]*.

Posteriormente, para cada linha e cada coluna da tabela verifica-se se o valor que se encontra nessa posição é menor ou maior, respectivamente, que os *min* e *max* considerados. No primeiro caso, se tal se verificar, então o valor *min* será actualizado, caso contrário continuará com o mesmo valor. No segundo caso, o processo é idêntico. Se a condição se verificar, o valor *max* será actualizado, caso contrário o valor *max* manter-se-á inalterado. Depois de se ter percorrido todas as linhas e todas as colunas, comparando os diferentes valores, segundo a ideia anteriormente referida, o valor *min* corresponderá ao menor valor que aparece na tabela e denotar-se-á por *lmin*, e o valor *max* corresponderá ao maior valor e denotar-se-á por *lmax*.

```
# lmin dá-nos o menor valor de l na tabela
# lmax dá-nos o maior valor de l na tabela

min = int(t[nl_nao][nc_nao])
max = int(t[nl_nao][nc_nao])
```

```
for i in range(nl_nao,nl):
    for j in range(nc_nao,nc):
        if (int(t[i][j]) < min):
            min = int(t[i][j])
        if (int(t[i][j]) > max):
            max = int(t[i][j])
lmin = min
lmax = max
#print "lmin =", lmin
#print "lmax =", lmax
```

No próximo passo é criado um novo dicionário, *obscoluna*, onde se armazena o número de vezes que cada valor da tabela é observado em cada coluna. Assim, para cada valor l, compreendido entre *lmin*, inclusive, e *lmax*, inclusive, verifica-se se esse valor ocupa uma determinada posição na tabela, isto é, uma determinada coluna e uma determinada linha. Em caso afirmativo, adiciona-se *1* à variável h, que inicialmente toma o valor *0*, para as diferentes colunas. Depois de percorridas todas as linhas, para cada uma das colunas, a variável h denota-se por *obscoluna[l,j]* e representa o número de vezes que o valor l se encontra na coluna j. É de notar que este processo se repete para todas as linhas, todas as colunas e todos os valores l possíveis na tabela fornecida.

Por exemplo,

$$obscoluna[9, 6] = 49.0$$

quer dizer que o valor 9 é observado 49 vezes na $7^a coluna$.

```
# obscoluna[l,j] é o dicionário que representa o número de vezes que o valor l é observado
na coluna j.
obscoluna = {}
for l in range( lmin , lmax+1 ):
```

```
#print "l =", l
for j in range( nc_nao , nc ):
    h = 0.0
    for i in range( nl_nao , nl ):
        if int ( t[i][j] ) == l:
            h = h + 1.0
    obscoluna[l,j] = h
    #print "obscoluna(", l , ",", j , ") =", obscoluna[l,j]
```

Para cada coluna do ficheiro de texto *YPortugal_ Total2.csv* é criada uma lista vazia, denominada por *lista*, na qual se armazenam os diferentes valores de l que aparecem nessa mesma coluna. Assim, para cada valor de l, começa-se por ver se esse valor aparece na coluna considerada, isto é, se $obscoluna[l, j] \neq 0.0$. Se tal condição se verificar, então adiciona-se *[l]* a *lista*. Depois de percorrer todos os valores de l, a lista *lista* contém todos os valores de l diferentes que aparecem na coluna j, e denominar-se-á *coluna[j]*. Estes valores são, posteriormente, armazenados no dicionário *coluna*.

Posteriormente é determinado o comprimento da lista *coluna[j]*, que nos dá o número de valores de l diferentes que aparecem na coluna j, e armazena-se essa informação na lista *valores[j]*. Estes valores são, posteriormente, guardados no dicionário *valores*.

Por exemplo,

$$coluna[2] = [13, 14, 15, 16, 17] \quad \text{e} \quad valores[2] = 5$$

isto é, 13,14,15,16 e 17 são os 5 valores diferentes que podem ser observados na $3^a coluna$.

```
# coluna[j] é a lista cujos elementos são os valores de l, diferentes, que se encontram na
coluna j.
# valores[j] dá-nos o número de valores de l diferentes na coluna j.

valores = {}
coluna = {}
```

```
for j in range(nc_nao,nc):
    lista = []
    for l in range(lmin,lmax+1):
        if obscoluna[l,j] != 0.0:
            lista = lista + [l]
    coluna[j] = lista
    #print "coluna[", j, "]=", coluna[j]
    valores[j] = len(coluna[j])
    #print "valores[", j, "]=", valores[j]
```

Com o intuito de garantir a execução eficiente do teste do *Qui-Quadrado* nos diferentes programas, começa-se por chamar o ficheiro de texto *Qui-Quadrado.csv*, o qual contém os dados necessários à realização dos diferentes passos, *g*.

```
# Chamar o ficheiro de texto Qui.csv, que contém os dados necessários ao teste do Qui-Quadrado.
g = open( "Qui-Quadrado.csv")
```

À semelhança do que foi efectuado, inicialmente, com o ficheiro de texto $YPortugal_Total2.csv$, é criado um ciclo, no qual se leêm as linhas do ficheiro *g*, e as quais se armazenam num dicionário denominado por *ss*. Em seguida, é criado um outro dicionário, *QQ*, no qual os espaços brancos que aparecem no dicionário *ss* são convertidos em vírgulas. Assim, cada *QQ[i]* representa a linha *i* do ficheiro, na qual os diferentes elementos estão separados por vírgulas, e cada elemento *QQ[i][j]* do dicionário *QQ*, representa o valor do ficheiro que se encontra na linha *i* e coluna *j*.

Por exemplo,

$$QQ[1] = ['1','2.7055','3.8415','5.0239','6.6349','7.8794']$$

$$QQ[1][2] = 3.8415$$

representam, repectivamente, a *2ª linha* do ficheiro *Qui-Quiadrado.csv* e o *3º elemento* dessa mesma linha.

As únicas diferenças, relativamente ao primeiro ficheiro, são a determinação do número de linhas e do número de colunas. Ao contrário do que acontece no ficheiro *YPortugal_Total2.csv*, neste ficheiro de texto, os dados referidos são conhecidos, *101* e *6*, respectivamente, e consequentemente, trabalha-se com o conhecimento dos mesmos.

```
# O dicionário ss representa as linhas do ficheiro.
# O dicionário QQ representa as mesmas linhas,mas sem os espaços em branco, estes são substituidos por vírgulas.
# QQ[i][j] representa o elemento que se encontra na linha i e coluna j.

ss = {}
QQ = {}
for i in range( 101 ):
    ss[i] = g.readline()
    QQ[i] = ss[i].split( )
    #for j in range( 6 ):
        #print "QQ(", i, ",", j, ") =", QQ[i][j]
```

No anexo - *Anexo C* - é possível visualizar um esquema do presente programa.

3.2 2colunas_Contiguidade.py

Neste programa pretende-se aplicar o teste do *Qui-Quadrado*, para determinar a dependência ou independência das colunas do ficheiro dado, duas a duas. Note-se, ainda, que na aplicação do teste referido se pretende ter em conta apenas valores esperados maiores

que *5*, pelo que se efectua o agrupamento dos diferentes valores que não obedecem à condição referida, tendo em conta, na junção, apenas valores **contíguos**.

Com o intuito de possibilitar o bom funcionamento do programa, começa-se por chamar e, posteriormente, executar o ficheiro criado anteriormente, *Dados.py*, o qual contém os dados necessários à execução do presente programa, *chamar*.

```
# Chamar o ficheiro Dados.py, que contém os dados necessários à execução do programa.

chamar = open("Dados.py")
exec chamar
```

De seguida, são determinadas o número de linhas observadas no ficheiro que têm um par ordenado de valores correspondentes a um par ordenado de colunas. Assim, para cada coluna, consideram-se as colunas posteriores, formando diferentes duplas com esta. Posteriormente, para cada duas colunas, consideram-se os valores possíveis nas mesmas e verifica-se se este par ordenado de valores, nas respectivas colunas, se verifica nas diferentes linhas. Caso esta condição seja satisfeita, adiciona-se *1* à variável h, que inicialmente toma o valor *0* para cada valor possivel na segunda coluna considerada em cada par ordenado de colunas. O valor h determinado é, posteriormente, armazenado na lista *dado[l,j,ll,jj]*, que nos dará o número de linhas observadas com os valores l e ll nas colunas j e jj, respectivamente.

Depois de todas as colunas serem consideradas, os diferentes valores calculados anteriormente são armazenados do dicionário *dado*.

É criado, também, um outro dicionário, *dadototal*, o qual contém os valores calculados anteriormente, tendo em conta o número total de linhas do ficheiro original. Assim, o dicionário *dadototal* armazena as diferentes listas *dadototal[l,j,ll,jj]*, onde cada uma delas é a razão entre o correspondente valor *dado[l,j,ll,jj]* e o número total de linhas do ficheiro em causa.

Por exemplo,

$$dado[14, 2, 13, 3] = 267.0 \qquad \text{e} \qquad dadototal[14, 2, 13, 3] = 0.406392694064$$

representam, repectivamente, o número de linhas observadas com os valores 14 e 13 nas 3ª e 4ª colunas, repectivamente, e o número de linhas observadas nas mesmas condições, mas tendo em conta o número total de linhas do ficheiro em estudo.

```
# dado[l,j,ll,jj] indica o número de linhas observadas com o valor l na coluna j e o valor ll
na coluna jj.
# dadototal[l,j,ll,jj] indica o número de linhas observadas com o valor l na coluna j e o
valor ll na coluna jj, tendo em conta o número total de linhas do ficheiro.

dado = {}
dadototal = {}
for j in range(nc_nao,nc):
    if j+1 < nc:
        for jj in range(j+1,nc):
            for l in coluna[j]:
                for ll in coluna[jj]:
                    h = 0
                    for i in range( nl_nao , nl ):
                        if ((int (t[i][j]) == l) and (int (t[i][jj]) == ll)):
                            h = h + 1.0
                    dado[l,j,ll,jj] = h
                    #print "dado[", l, ",", j , ",", ll, ",", jj , "] =", dado[l,j,ll,jj]
                    dadototal[l,j,ll,jj] = dado[l,j,ll,jj] / (nl-nl_nao)
                    #print "dadototal[", l, ",", j , ",", ll, ",", jj , "] =", dadototal[l,j,ll,jj]
```

Em seguida, é calculada a probabilidade de um determinado valor se encontrar numa determinada coluna do ficheiro em causa. Sendo assim, para cada coluna consideram-se os diferentes valores possíveis na mesma, e a probabilidade de ter um valor l na coluna j, *probcoluna[l,j]*, é a razão entre o número de vezes que l se encontra em j, isto

é, *obscoluna[l,j]*, e o número total de linhas do ficheiro. Cada uma das probabilidades calculadas é armazenada no dicionário *probcoluna*.

Por exemplo,

$$probcoluna[13, 2] = 0.127853881279$$

significa que a probabilidade de ter o valor 13 na $3^a coluna$ é 0.127853881279.

```
# probcoluna[l,j] representa a probabilidade de ter o valor l na coluna j.

probcoluna = {}
for j in range(nc_nao,nc):
    for l in coluna[j]:
        probcoluna[l,j] = obscoluna[l,j]/(nl-nl_nao)
        #print "probcoluna[", l, ",", j, "] =", probcoluna[l,j]
```

No próximo passo, seguindo um raciocínio análogo ao anteriormente referido, consideram-se pares de colunas e os respectivos valores possíveis nas mesmas. Note-se que, para cada dupla de colunas, é considerada um variável auxiliar, i, inicialmente nula, e que indica o número de pares de valores diferentes associados ao par de colunas consideradas. São, também, criados dois dicionários, D e Q. No primeiro são armazenadas as listas *D[i]*, que representam os diferentes pares da forma *[l,j,ll,jj]*, isto é, encontrar os valores l e ll nas colunas j e jj, respectivamente. O dicionário Q, armazena as listas *Q[i]*, as quais estão associadas à respectiva lista *D[i]* e, fornecem-nos o produto das probabilidades de ter os valores l e ll nas colunas j e jj, respectivamente, pelo número total de linhas. Note-se que os valores armazenados no dicionário Q correspondem ao número esperado de linhas do ficheiro de texto, com os pares de colunas e valores correspondentes, da forma daqueles que são armazenados no dicionário D, admitindo independência.

Note-se, ainda, que o valor i, que nos indica o i-ésimo valor dos dicionários D e Q, para o par de colunas que está a ser considerado, indica-nos também os valores do domínio dos mesmos, isto é, os valores i que compõem os dominios de D e Q.

Por exemplo,

$$D[2] = [13, 2, 10, 3] \quad e \quad Q[2] = 0.127853881279$$

significa que o 3^o*elemento* do domínio de D é $[13, 2, 10, 3]$, associado ao 3^o*elemento* do domínio de Q, que é 0.127853881279, significando o número esperado de linhas do ficheiro em estudo, com os valores 13 e 10, respectivamente, nas 3^a e 4^a colunas.

```
# D[i] é a lista que representa o i-ésimo par da forma [l,j,ll,jj], isto é, valores l e ll nas colunas j e jj, respectivamente.
# Q[i] é a lista que representa o número de linhas esperado com o par da forma [l,j,ll,jj].
# cont[x,y] é a lista que indica se D[x] e D[y] são contíguos (= 1) ou não (= 0).
# Listas D e Q, após a junção dos pequeninos contíguos entre si e junção dos pequeninos contíguos, ao grande mais próximo.
# Qui[j,jj] representa a estatística do teste Qui-Quadrado, para as colunas j e jj.
# gl[j,jj] dá-nos os graus de liberdade entre as colunas j e jj.

g2c = {}
for j in range(nc_nao,nc):
    if (j+1 < nc):
        for jj in range(j+1,nc):
            i = 0
            D = {}
            Q = {}
            for l in coluna[j]:
                for ll in coluna[jj]:
                    D[i] = [l,j,ll,jj]
                    Q[i] = probcoluna[l,j]*probcoluna[ll,jj]*(nl-nl_nao)
                    i = i + 1
```

É de notar que nos passos posteriores, e tendo em conta que não é aconselhável considerar valores esperados menores que 5, agrupam-se os valores do domínio, contíguos, com valor esperado menor que 5, por ordem crescente, até serem maiores que *5*. No caso de não dar para agrupar mais os valores, e se continuar a ter valores esperados menores que *5*, então estes serão agrupados, por ordem decrescente, ao valor do dominio mais próximo com valor esperado maior que *5*. Os passos referidos serão descritos posteriormente.

Seguindo o ciclo anterior, para cada par de colunas é considerado um dicionário, *cont*, inicialmente vazio, e no qual se pretende armazenar o valor *1*, no caso dos valores serem contíguos, e *0*, caso contrário, e uma variável auxiliar w, inicialmente nula. Posteriormente, inicia-se um ciclo, o qual será percorrido enquanto esta variável continuar a tomar o valor *0*, caso contrário, o ciclo será interrompido. Assim, quando se inicia o ciclo, a variável w passa a tomar o valor *1*, e para cada elemento do domínio de D, que se denota por z, consideram-se mais duas variáveis auxiliares, C e ww inicialmente, também, nulas. Seguidamente, e como cada par da forma *[l,j,ll,jj]* tem comprimento *4*, percorrem-se os elementos da lista *D[z]* (note-se que z é o elemento do dominio de D que se tomou anteriormente), a qual, inicialmente, tem comprimento *4*, também, em passos de *4*, e chama-se a esta variável, p. Assim, começa-se por ver se a variável C toma valor *1* (condição que no primeiro passo será avançada, visto a variável ser *0*, de inicio.), caso em que o ciclo será automaticamente interrompido. Caso contrário, consideram-se duas variáveis, k e h. A primeira toma valores entre *2*, inclusive, e *8*, exclusive, e a segunda entre *0* exclusive e $\frac{k}{2}$, inclusive. De seguida, se o valor *Q[z]*, valor de Q para o elemento do domínio considerado, for menor que o valor de h que se está a considerar, então prossegue-se, considerando um outro valor do domínio de D, t, diferente do anterior. Para este valor, à semelhança do que foi feito, anteriormente, para z, percorrem-se os elementos da lista *D[t]* em passos de 4, denotando a variável em causa por pp, e faz-se o cálculo:

$$(D[z][p] - D[t][pp])^2 + (D[z][p+2] - D[t][pp+2])^2$$

Note-se que para cada par da forma *[l,j,ll,jj]* os valores que são considerados no cálculo anterior são respectivamente, l e ll.

Caso este valor seja igual a *1*, então diz-se que as listas *D[z]* e *D[t]* são contíguas, e o valor C passa a ser igual a *1*, e o ciclo é interrompido, caso contrário, as listas não são

contíguas e o valor C toma o valor *0*. Neste caso, passa-se ao próximo valor de *pp*, até encontrar um par de valores do dominio de D contíguos, isto é, com $C = 1$, caso esse valor exista. Seguidamente, ao valor C chama-se *cont[z,t]*, valor que é armazenado no dicionário *cont*.

```
cont = {}
w = 0
while (w == 0):
    w = 1
    for z in D.keys():
        C = 0
        ww = 0
        for p in range(0,len(D[z]),4):
            if (C == 1):
                break
            else:
                for k in range(2,8):
                    for h in range(int(k/2),0,-1):
                        if (Q[z] < h):
                            for t in D.keys():
                                if (t != z):
                                    for pp in range(0,len(D[t]),4):
                                        if (((D[z][p]-D[t][pp])**2)+((D[z][p+2]-D[t][pp+2])**2)==1):
                                            C = 1 # ié, D[z] contíguo com D[t]
                                            break
                                        else:
                                            C = 0
                                    cont[z,t] = C
```

Posteriormente, verifica-se se o valor calculado anteriormente é igual a *1*, caso em que z e t são contíguos, e se o valor de Q para o elemento t do dominio, isto é, *Q[t]*, é menor que $k - h$. Caso estas duas condições sejam satisfeitas, então a variável w passa a ser novamente *0*, e o t-ésimo valor do dominio de D e de Q, respectivamente, são agrupados com o z-ésimo valor dos mesmos dominios, sendo, seguidamente, eliminados os t-ésimos valores, isolados. A variável *ww* passa a ser 1, e o ciclo é interrompido.

Em seguida, ainda dentro do ciclo do elemento z do domínio de D, verifica-se se a variável *ww* tem o valor *1*. Caso seja satisfeita esta condição, então o ciclo é interrompido, caso contrário, passa-se ao próximo elemento z.

```
                    if ((cont[z,t] == 1) and (Q[t] < k-h)):
                        w = 0
                        D[z] = D[z] + D[t]
                        Q[z] = Q[z] + Q[t]
                        del D[t]
                        del Q[t]
                        ww = 1
                        break
            if (ww == 1):
                break
    #print "D =", D
    #print "Q =", Q
```

Por exemplo, se $k = 6$ e $h = 3$,

$$cont[5, 11] = 1$$

significa que os valores 5 e 11 do domínio de D, dados por $D[5] = [13, 2, 15, 3]$ e $D[11] = [14, 2, 15, 3]$ são contíguos e, nestas condições, sendo $Q[5] < 6$ e $Q[11] < 3$, então os valores 5 e 11 dos domínios de D e Q vaõ ser agrupados, obtendo-se:

$$D[5] = [13, 2, 15, 3, 14, 2, 15, 3] \qquad e \qquad Q[5] = 3.6377473363774726$$

Nos passos anteriores, foram agrupados os valores do domínio, contíguos, com valor esperado menor que *5*, por ordem crescente e, posteriormente, vão ser agrupados aqueles com valor esperado menor que *5*, ao contíguo com valor esperado maior que *5*.

Considera-se, então, uma variável, k que toma valores entre *0*, exclusive, e *5*, inclusive. De seguida, considera-se uma variável auxiliar, H, inicialmente nula, e inicia-se um ciclo, o qual será interrompido, quando a variável H deixar de ser nula. Assim, enquanto essa variável continuar a ser igual a *0*, no início do ciclo, altera-se o seu valor para *1*, e para cada valor do dominio do dicionário D, denotado por x, consideram-se duas variáveis auxiliares, C e *ww*, inicialmente nulas. À semelhança do que foi feito anteriormente, percorrem-se os elementos da lista *D[x]* em passos de 4, e chama-se a esta variável, p. De seguida, começa-se por ver se a variável C toma valor *1* (condição que no primeiro passo é avançada, visto a variável ser *0*, de inicio). Neste caso, o ciclo é automaticamente interrompido. Caso contrário, verifica-se se o valor *Q[x]*, é menor que k e maior ou igual que *k-1*. No caso destas duas condições se verificarem, considera-se uma variável auxiliar z, inicialmente igual a *99*, (maior valor possível com *2* dígitos - Note-se que os valores do ficheiro de texto *YPortugal_Total2.csv* têm *1* ou *2* digitos) e um outro elemento do domínio, y, diferente do anterior, para o qual se percorrem os elementos da lista *D[y]*, em passos de *4*, denotando a variável em causa por *pp* e, análogo ao que foi feito anteriormente, faz-se o cálculo:

$$(D[x][p] - D[y][pp])^2 + (D[xx][p+2] - D[y][pp+2])^2$$

Caso este valor seja igual a *1*, então diz-se que as listas *D[x]* e *D[y]* são contíguas, e o valor C passa a ser igual a *1*, e o ciclo de *pp* é interrompido, caso contrário, as listas não são contíguas e o valor C toma o valor *0*. Neste caso, analogamente ao que foi feito anteriormente, passa-se ao próximo valor *pp* do domínio, até encontrar um valor y contíguo com x, isto é, em que $C = 1$, caso esse valor exista. Seguidamente, de forma análoga ao que foi feito nos passo descrito anteriormente, ao valor C chama-se *cont[z,t]*, valor que é armazenado no dicionário *cont*.

Posteriormente, verifica-se se o valor calculado anteriormente é igual a *1*. Neste caso, no primeiro passo, isto é, quando a variável z considerada anteriormente, toma o valor *99*, faz-se z igual a y, isto é, a variável z passa a tomar um novo valor, y. Caso contrário, isto é, depois do primeiro passo, verifica-se se o valor esperado do elemento y do domínio é menor que o do valor z, isto é, se $Q[y] < Q[z]$. Caso esta condição se verifique, faz-se z igual a y.

Posteriormente, tendo ainda em conta que o valor *cont[z,t]* é *1*, o y-ésimo valor do dominio de D e de Q, respectivamente, são agrupados com o x-ésimo valor dos mesmos dominios, sendo, seguidamente, eliminados os y-ésimos valores, isolados. A variável H passa a ser *0* e a *ww* passa a ser 1, e o ciclo é interrompido.

Em seguida, ainda dentro do ciclo do elemento x do domínio de D, verifica-se se a variável *ww* tem o valor *1*. Caso seja satisfeita esta condição, então o ciclo é interrompido, caso contrário, passa-se ao próximo elemento x.

```
for k in range(5,0,-1):
    H = 0
    while (H == 0):
        H = 1
        for x in D.keys():
            C = 0
            ww = 0
            for p in range(0,len(D[x]),4):
                if (C == 1):
                    break
                else:
                    if ((Q[x] < k) and (Q[x] >= k-1)):
                        z = 99
                        for y in D.keys():
                            if (y != x):
                                for pp in range(0,len(D[y]),4):
```

```
                    if (((D[x][p]-D[y][pp])**2)+((D[x][p+2]-D[y][pp+2])**2)==
1):
                        C = 1 # ié, D[x] contíguo com D[y]
                        break
                    else:
                        C = 0
                cont[x,y] = C
                if (cont[x,y] == 1):
                    if (z == 99):
                        z = y
                    else:
                        if (Q[y] < Q[z]):
                            z = y
                    D[x] = D[x] + D[y]
                    Q[x] = Q[x] + Q[y]
                    del D[y]
                    del Q[y]
                    H = 0
                    ww = 1
                    break
            if (ww == 1):
                break
        #print "D =", D
        #print "Q =", Q
```

No próximo passo é feita uma representação, para cada duas colunas, com o intuito de possibilitar uma visualização dos valores agrupados. Assim, para cada valor de cada uma das colunas consideradas, x e y, respectivamente, começa-se por tomar um valor do domínio de D, z, e uma variável, g, inicialmente nula. Posteriormente, para cada valor

z percorrem-se os valores de *D[z]* em passos de *4*, visto cada par da forma *[l,j,ll,jj]* ter comprimento *4*, denotando-se esta variável por *w*. De seguida, verifica-se se os valores x e y correspondentes à primeira e segunda colunas consideradas, respectivamente, são iguais a *D[z][w]* e *D[z][w+2]*, isto é, ao primeiro (posição *0*) e terceiro (posição *2*) valor de cada lista *[l,j,ll,jj]*, respectivamente. Neste caso, representa-se o valor do dominio sob a forma *z+10*, com o objectivo deste ser representado com *2* digitos, e a variável *g* passa a tomar o valor *1*, e o ciclo é interrompido. Note-se que, os valores de cada coluna, na representação estão separados por uma coluna ou linha.

```
for x in coluna[j]:
   for y in coluna[jj]:
      for z in D.keys():
         g = 0
         for w in range(0,len(D[z])-3,4):
            if ((x == D[z][w]) and (y == D[z][w+2])):
               #print z+10,
               g = 1
               break
         if g == 1:
            break
   #print
#print
```

Por exemplo, para as colunas *2* (indicada por cada linha) e *3* (indicada por cada coluna) a representação seria:

34 34 12 13 14 27

34 34 18 19 20 27

34 34 24 25 27 27

34 34 34 31 32 27

34 34 38 38 38 38

onde o facto das posições apresentarem o mesmo valor indica que as mesmas se encontram agrupadas.

Em seguida, consideram-se três listas, inicialmente vazias, para cada par de colunas, *obs*, *Qui*, *gl*, que armazenam as listas *obs[x]*, *Qui[j,jj]* e *gl[j,jj]*, respectivamente, que indicam, o número de vezes que o x-ésimo elemento do domínio de D é observado no ficheiro original, o valor do *Qui-Quadrado* e os graus de liberdade, calculados para cada par de colunas. É, ainda, considerada uma variável, à priori nula, *var*.

Posteriormente, para cada elemento do domínio do dicionário D, x, calcula-se o número de vezes que este elemento é observado no ficheiro dado, isto é, percorrendo os elementos da lista *D[x]* em passos de *4*, pelo motivo já referido anteriormente, e denotando a variável por p, ao valor *obs[x]*, inicialmente nulo, adiciona-se o valor *dado[l,j,ll,jj]*, onde l é agora *D[x][p]* e *ll D[x][p+2]*, já calculados anteriormente. Note-se, tal como foi feito com os dicionários D e Q, anteriormente, agora também se está a actualizar o dicionário *dado*, denotando-o, desta vez, por *obs*.

Por exemplo,

$$obs[10] = 17.0$$

representa o número observado de linhas do ficheiro em causa, nas condições associadas à lista $D[10] = [20, 15, 29, 16, 20, 15, 28, 16]$

Seguidamente, é determinado o valor do Qui-Quadrado, o qual se determina da seguinte forma:

$$\sum_{x \in D.keys} \frac{(obs[x] - Q[x])^2}{Q[x]}$$

pelo que, para cada x do domínio de D, se vai adicionando a *var* a parcela $\frac{(obs[x]-Q[x])^2}{Q[x]}$. Depois de calculado o valor de *var* para todos os valores x do domínio de D, denota-se o valor obtido por *Qui[j,jj]*, uma vez que representa o valor do *Qui-Quadrado*, calculado para as colunas j e jj.

Por exemplo,

$$Qui[2, 3] = 37.9204286735$$

é o valor do *Qui-Quadrado* calculado, associado às 3ª e 4ª colunas (mais uma vez, note-se que em *Python* a contagem começa em 0.)

No passo que se segue, determinam-se os graus de liberdade referente às colunas em causa, o qual corresponde ao número de valores do domínio do dicionário D menos uma unidade.

Por exemplo,

$$gl[2, 3] = 13 - 1 = 12$$

sendo 12 os graus de liberdade associados às colunas 2 e 3.

Finalmente, é realizado o teste do *Qui-Quadrado*, o qual consiste na determinação da dependência ou independência do par de colunas que estão a ser consideradas, comparando o valor do *Qui-Quadrado* calculado com o tabelado. No caso do primeiro ser maior ou igual que o segundo, as colunas em causa dizem-se dependentes e o valor armazenado no dicionário *g2c*, correspondente ao par de colunas em causa, é *1*. Caso contrário, o valor armazenado é *0*, e as colunas dizem-se independentes.

Considerando as colunas 2 e 3, como anteriormente, o valor do *Qui-Quadrado* tabelado e associado a essas colunas é

$$\chi^2_{12,0.05} = 21.0261$$

Como este valor é menor que o anterior, concluir-se-ia que as colunas mencionadas eram *dependentes.*)

```
obs = {}
Qui = {}
var = 0.0
gl = {}
for x in D.keys():
    obs[x] = 0
    for p in range(0,len(D[x]),4):
        obs[x] = obs[x] + dado[D[x][p],j,D[x][p+2],jj]
    var = var + (((obs[x]-Q[x])**2)/Q[x])
Qui[j,jj] = var
#print "Qui[", j , ",", jj, "] =", Qui[j,jj]
gl[j,jj] = int(len(D.keys())-1)
#print "gl[", j, ",", jj, "] =", gl[j,jj]
# Teste do Qui-Quadrado - verifica se 2 colunas são ou não dependentes:
#print "QQ[", gl[j,jj], "][2] =", float(QQ[gl[j,jj]][2])
if (Qui[j,jj] >= float(QQ[gl[j,jj]][2])):
    g2c[j,jj] = 1
    print "As colunas", j, "e", jj, "são dependentes"
else:
    g2c[j,jj] = 0
    print "As colunas", j, "e", jj, "são independentes"
#print "g2c =", g2c
```

Note-se que, no valor do *Qui-Quadrado* tabelado, como já vimos anteriormente no programa *dado.py*, se encontra armazenado no dicionário *QQ*, apenas interessa a linha

correspondente aos graus de liberdade calculados e segunda coluna, isto é, *QQ[gl[j,jj]][2]*. Salienta-se que tal coluna é a considerada visto tratar-se da coluna que armazena os valores do *Qui-Quadrado* referentes à probabilidade *0.05*, e os graus de liberdade possiveis vão de *0* a *100*, inclusive.

Ao executar o programa descrito, no que diz respeito a casos de independência, os resultados obtidos são os seguintes:

colunas 2 *e* 13, *colunas* 2 *e* 16, *colunas* 3 *e* 4, *colunas* 3 *e* 13, *colunas* 4 *e* 11, *colunas* 4 *e* 12, *colunas* 4 *e* 16, *colunas* 5 *e* 12, *colunas* 5 *e* 13, *colunas* 6 *e* 13, *colunas* 8 *e* 11, *colunas* 8 *e* 12, *colunas* 8 *e* 13, *colunas* 11 *e* 13, *colunas* 11 *e* 16, *colunas* 12 *e* 13, *colunas* 13 *e* 14, *colunas* 13 *e* 15, e *colunas* 13 *e* 16

Os restantes *86* pares de colunas, não mencionados, são dependentes.

Note-se que, quando se diz que o par de colunas 2 *e* 13 é independente, em termos de genética, pretende-se dizer que há ausência de associação entre os marcadores presentes nas colunas 2 *e* 13 do ficheiro *YPortugal_Total2.csv*.

Convém, ainda, referir que no *excel*, à primeira coluna se associa a posição *0*, à segunda a posição *1* e assim sucessivamente, passando-se exactamente o mesmo em relação às linhas. Este facto é tido em conta com o ficheiro fornecido pelo *IPATIMUP*, *YPortugal_Total2.csv*, na execução dos diferentes programas elaborados.

3.3 3colunas_Contiguidade.py

O presente programa é bastante semelhante ao programa anterior, agrupando-se, também, os valores **contíguos** com valor esperado menor que *5*. Salienta-se que a diferença nos passos seguidos se deve ao facto de se considerarem três colunas, em vez de duas. Tendo em conta a semelhança no raciocínio dos passos seguidos, os mesmos não serão

pormenorizadamente descritos, não se justificando, também, a apresentação de exemplos ao longo do programa.

À semelhança do que foi feito no programa *2colunas_Contiguidade.py*, neste programa começa-se por chamar e, posteriormente, executar o ficheiro *Dados.py*, com os dados essenciais à execução do mesmo, *chamar*.

```
# Chamar o ficheiro Dados.py, que contém os dados necessários à execução do programa.
chamar = open("Dados.py")
exec chamar
```

De seguida, é determinado o número de linhas observadas no ficheiro que têm um conjunto ordenado de valores correspondentes a um conjunto ordenado de colunas. No programa já referido, este procedimento é efectuado para dois valores e duas colunas. No presente programa, segue-se um raciocínio análogo, mas para três valores e três colunas. Assim, para cada coluna consideram-se, apenas, as colunas posteriores, formando diferentes triplas. Posteriormente, para cada três colunas, consideram-se os valores possíveis nas mesmas e verifica-se se a tripla ordenada de valores, nas respectivas colunas, se verifica nas diferentes linhas. Caso esta condição seja satisfeita, da mesma forma que no programa *2colunas_Contiguidade.py*, adiciona-se *1* à variável auxiliar *h*, que inicialmente toma o valor *0*, para cada valor possivel na terceira coluna considerada em cada tripla ordenada de colunas. O valor *h* determinado é, posteriormente, armazenado na lista *dado[l,j,ll,jj,lll,jjj]*, que nos dará o número de linhas observadas com os valores *l*, *ll* e *lll* nas colunas *j*, *jj* e *jjj* respectivamente. Depois de todas as colunas serem consideradas, os diferentes valores calculados anteriormente são armazenados no dicionário *dado*.

Analogamente ao programa referido, é criado um outro dicionário, *dadototal*, que contém os valores calculados anteriormente, tendo em conta o número total de linhas do ficheiro original. Tem-se, portanto, que o dicionário *dadototal* armazena as diferentes listas *dadototal[l,j,ll,jj,lll,jjj]*, onde cada uma delas é a razão entre o correspondente valor *dado[l,j,ll,jj,lll,jjj]* e o número total de linhas do ficheiro em causa.

dado[j,l,jj,ll,jjj,lll] indica o número de linhas observadas com os valores l, ll e lll nas colunas j, jj e jjj, respectivamente.
dadototal[j,l,jj,ll,jjj,lll] indica o número de linhas observadas com os valores l, ll e lll nas colunas j, jj e jjj, respectivamente, tendo em conta o número total de linhas do ficheiro.

```
dado = {}
dadototal = {}
for j in range(nc_nao,nc):
    if j+1 < nc:
        for jj in range(j+1,nc):
            if jj+1 < nc:
                for jjj in range(jj+1,nc):
                    for l in coluna[j]:
                        for ll in coluna[jj]:
                            for lll in coluna[jjj]:
                                h = 0
                                for i in range( nl_nao , nl ):
                                    if ((int (t[i][j]) == l) and (int (t[i][jj]) == ll) and (int (t[i][jjj])==
lll)):
                                        h = h + 1.0
                                dado[l,j,ll,jj,lll,jjj] = h
                                #print "dado[", l, ",", j, ",", ll, ",", jj, ",", lll, ",", jjj, "] =",
                                dado[l,j,ll,jj,lll,jjj]
                                dadototal[l,j,ll,jj,lll,jjj] = dado[l,j,ll,jj,lll,jjj] / (nl-nl_nao)
                                #print "dadototal[", l, ",", j, ",", ll, ",", jj, ",", lll, ",", jjj, "] =",
                                dadototal[l,j,ll,jj,lll,jjj]
```

Em seguida, é calculada a probabilidade de um determinado valor, *l*, se encontrar numa determinada coluna, *j*, do ficheiro de texto em causa, *probcoluna[l,j]*, sendo cada uma delas armazenada no dicionário *probcoluna*. O processo seguido para a sua determinação

é exactamente igual ao descrito no programa mencionado anteriormente.

```
# probcoluna[l,j] representa a probabilidade de ter o valor l na coluna j.

probcoluna = {}
for j in range(nc_nao,nc):
    for l in coluna[j]:
        probcoluna[l,j] = obscoluna[l,j]/(nl-nl_nao)
        #print "probcoluna[", l, ",", j, "] =", probcoluna[l,j]
```

Nos próximos passos, tal como nos precedentes, segue-se o mesmo raciocínio do programa *2colunas_Contiguidade.py*, pelo que os passos não serão descritos pormenorizadamente, sendo apenas referidas as diferenças.

Neste programa, em vez de se considerarem duplas de colunas, consideram-se triplas de colunas e os respectivos valores possíveis nas mesmas.

Os valores armazenados nas listas *D[i]* que, posteriormente, são guardados no dicionário *D*, representam os diferentes pares da forma *[l,j,ll,jj,lll,jjj]*, que corresponde a encontrar os valores *l*, *ll* e *lll* nas colunas *j*, *jj* e *jjj*, respectivamente. Os valores armazenados no dicionário *Q*, listas *Q[i]*, correspondem ao número esperado de linhas do ficheiro de texto, com as triplas de colunas e valores correspondentes, da forma daqueles que são armazenados no dicionário *D*, admitindo independência, e são determinados da forma descrita no programa anterior.

```
# D[i] é a lista que representa o i-ésimo par da forma [l,j,ll,jj,lll,jjj], ié, valores l e ll nas colunas j e jj, resp.
# Q[i] é a lista que representa o número de linhas esperado com o par da forma [l,j,ll,jj,lll,jjj].
# cont[x,y] é a lista que indica se D[x] e D[y] são contíguos (= 1) ou não (= 0).
# Listas D e Q, após a junção dos pequeninos contíguos entre si e junção dos pequeninos contíguos, ao grande mais próximo.
# Qui[j,jj,jjj] representa a estatística do teste Qui-Quadrado, para as colunas j e jj.
```

```
# gl[j,jj,jjj] dá-nos os graus de liberdade entre as colunas j e jj.

g3c = {}
for j in range(nc_nao,nc):
    if (j+1 < nc):
        for jj in range(j+1,nc):
            if (jj+1 < nc):
                for jjj in range(jj+1,nc):
                    i = 0
                    D = {}
                    Q = {}
                    for l in coluna[j]:
                        for ll in coluna[jj]:
                            for lll in coluna[jjj]:
                                D[i] = [l,j,ll,jj,lll,jjj]
                                Q[i] = probcoluna[l,j]*probcoluna[ll,jj]*probcoluna[lll,jjj]*(nl-nl_nao)
                                i = i + 1
```

Nos passos seguintes, agrupam-se os valores do domínio, contíguos, com valor esperado menor que *5*, por ordem crescente.

Note-se que cada par da forma *[l,j,ll,jj,lll,jjj]* tem comprimento *6*, pelo que se percorrem os elementos da lista *D[z]* e da lista *D[t]*, sob as condições descritas no programa *2colunas_Contiguidade.py*, onde z e t são elementos, diferentes, do domínio de D, em passos de *6*, chamando-se a estas variáveis, p e pp, respectivamente. Para estes valores do domínio faz-se o cálculo:

$$(D[z][p] - D[t][pp])^2 + (D[z][p+2] - D[t][pp+2])^2 + (D[z][p+4] - D[t][pp+4])^2$$

De forma análoga ao programa anterior, em cada par da forma *[l,j,ll,jj,lll,jjj]*, os valores que são considerados no cálculo anterior são respectivamente, l, ll e lll.

Caso este valor seja igual a *1*, então diz-se que as listas *D[z]* e *D[t]* são contíguas, caso contrário, as listas não são contíguas. A este valor chama-se *cont[z,t]*, e armazena-se no dicionário *cont*.

Nas mesmas condições do programa anterior, o t-ésimo valor do dominio de D e de Q, respectivamente, são agrupados com o z-ésimo valor dos mesmos dominios, e os t-ésimos valores isolados são, seguidamente, eliminados.

```
cont = {}
w = 0
while (w == 0):
    w = 1
    for z in D.keys():
        C = 0
        ww = 0
        for p in range(0,len(D[z]),6):
            if (C == 1):
                break
            else:
                for k in range(2,8):
                    for h in range(int(k/2),0,-1):
                        if (Q[z] < h):
                            for t in D.keys():
                                if (t != z):
                                    for pp in range(0,len(D[t]),6):
                                        if (((D[z][p]-D[t][pp])**2)+((D[z][p+2]-D[t][pp+2])**2)
+((D[z][p+4]-D[t][pp+4])**2)==1):
                                            C = 1 # ié, D[x] contíguo com D[y]
                                            break
                                        else:
```

```
                    C = 0
                cont[z,t] = C
                if ((cont[z,t] == 1) and (Q[t] < k-h)):
                    w = 0
                    D[z] = D[z] + D[t]
                    Q[z] = Q[z] + Q[t]
                    del D[t]
                    del Q[t]
                    ww = 1
                    break
            if (ww == 1):
                break
        #print "D =", D
        #print "Q =", Q
```

Nos passos posteriores, agrupam-se, por ordem decrescente, os valores dos domínios de D e Q, com valor esperado menor que *5*, ao contíguo com valor esperado maior que *5*.

De forma idêntica ao passo anterior, percorrem-se os elementos da lista *D[x]* e da lista *D[y]*, sob as mesmas condições consideradas no programa descrito anteriormente, onde x e y são elementos, diferentes, do domínio de D, em passos de *6*, chamando-se a estas variáveis, p e pp, respectivamente. Para estes valores do domínio faz-se o cálculo:

$$(D[x][p] - D[y][pp])^2 + (D[x][p+2] - D[y][pp+2])^2 + (D[x][p+4] - D[y][pp+4])^2$$

Se este valor for igual a *1*, caso em que as listas *D[x]* e *D[y]* são contíguas, procede-se de forma análogo ao programa anterior, passando a variável z a tomar um novo valor, y, quando o valor z é igual a *99*, ou quando o valor esperado do elemento y do domínio é menor que o do valor z, isto é, $Q[y] < Q[z]$.

Posteriormente, tendo ainda em conta que o valor *cont[z,t]* é *1*, o y-ésimo valor do dominio de D e de Q, respectivamente, são agrupados com o x-ésimo valor dos mesmos dominios, sendo, de seguida, eliminados os y-ésimos valores, isolados.

```
for k in range(5,0,-1):
    H = 0
    while (H == 0):
        H = 1
        for x in D.keys():
            C = 0
            ww = 0
            for p in range(0,len(D[x]),6):
                if (C == 1):
                    break
                else:
                    if ((Q[x] < k) and (Q[x] >= k-1)):
                        z = 99
                        for y in D.keys():
                            if (y != x):
                                for pp in range(0,len(D[y]),6):
                                    if (((D[x][p]-D[y][pp])**2)+((D[x][p+2]-D[y][pp+2])**2)+
((D[x][p+4]-D[y][pp+4])**2)==1):
                                        C = 1 # ié, D[x] contíguo com D[y]
                                        break
                                    else:
                                        C = 0
                                cont[x,y] = C
                                if (cont[x,y] == 1):
                                    if (z == 99):
```

```
                    z = y
                else:
                    if (Q[y] < Q[z]):
                        z = y
                D[x] = D[x] + D[y]
                Q[x] = Q[x] + Q[y]
                del D[y]
                del Q[y]
                H = 0
                ww = 1
                break
        if (ww == 1):
            break
    #print "D =", D
    #print "Q =", Q
```

Com o objectivo de possibilitar uma visualização dos valores agrupados, no próximo passo é feita uma representação, a três dimensões, para cada três colunas. Assim, para cada valor de cada uma das colunas consideradas, x, y e t, respectivamente, toma-se um valor do domínio de D, z, para o qual se percorrem os valores de *D[z]* em passos de *6*. Posteriormente, verifica-se se os valores x, y e t são iguais, respectivamente, a *D[z][w]*, *D[z][w+2]* e *D[z][w+4]*, isto é, ao primeiro (posição *0*), terceiro (posição *2*) e quinto (posição *4*) valor de cada lista *[l,j,ll,jj,lll,jjj]*. Neste caso, representa-se o valor do dominio sob a forma *z+100* (com *3* dígitos).

```
for x in coluna[j]:
    for y in coluna[jj]:
        for t in coluna[jjj]:
            for z in D.keys():
```

```
            g = 0
            for w in range(0,len(D[z])-5,6):
                if ((x == D[z][w]) and (y == D[z][w+2]) and (t == D[z][w+4])):
                    #print z+100,
                    g = 1
                    break
            if g == 1:
                break
        #print
    #print
```

Por exemplo, para as colunas *2*, *3* e *4* a representação seria:

152 152 152 152 152 152

152 152 152 152 152 152

152 152 115 115 152 207

152 269 120 127 127 207

269 269 126 127 207 207

269 269 200 200 207 207

152 152 152 152 152 152

152 152 152 152 152 207

152 149 149 151 152 207

182 155 156 157 158 207

269 197 162 163 164 207

269 269 200 200 200 207

152 152 182 182 152 207

152 182 182 152 182 207

182 182 186 223 182 273

182 191 192 229 194 273

182 197 198 199 200 207

269 197 273 273 200 207

152 182 182 182 182 207

252 182 182 182 182 182

252 252 252 223 182 273

182 252 228 229 273 273

269 197 273 273 273 273

269 273 273 273 273 207

252 252 182 182 273 273

252 252 252 252 182 273

252 252 252 223 273 273

269 269 273 273 273 273

269 269 269 273 273 273

269 269 273 273 273 273

onde, da mesma forma que no programa anterior, o facto das posições apresentarem o mesmo valor indica que as mesmas se encontram agrupadas. Note-se que este conjunto de matrizes corresponde a uma representação tridimensional do espaço dos valores: os valores

da coluna *2* correspondem ao "comprimento", os valores da coluna *3* correspondem à "largura"e os valores da coluna *4* correspondem à "altura"(traduzida na existência dos diversos planos).

Seguidamente, para cada elemento do domínio do dicionário D, x, percorrem-se os elementos da lista *D[x]* em passos de *6*, denotando a variável por p, e calcula-se o número de vezes que este elemento é observado no ficheiro dado, *obs[x]*. À semelhança do que foi efectuado no programa anterior, adiciona-se ao valor *obs[x]*, à priori nulo, o valor *dado[l,j,ll,jj,lll,jjj]*, onde *l* é agora *D[x][p]*, *ll* é *D[x][p+2]* e *lll* é *D[x][p+4]*. Este novo dicionário, *obs*, corresponde à actualização do dicionário *dado*.

Posteriormente, determina-se o valor do Qui-Quadrado para cada tripla de colunas, *Qui[j,jj,jjj]*, que se determina da mesma forma que no programa anterior:

$$\sum_{x \in D.keys} \frac{(obs[x] - Q[x])^2}{Q[x]}$$

Os graus de liberdade referentes à tripla de colunas em causa, *gl[j,jj,jjj]*, correspondem ao número de valores do domínio do dicionário D menos uma unidade, tal como anteriormente.

Para terminar, realiza-se o teste do *Qui-Quadrado*, de forma exactamente igual à efectuada no programa anterior, comparando o valor do *Qui-Quadrado* calculado com o tabelado, concluindo dependência das colunas no caso do primeiro ser maior ou igual que o segundo, e independência, caso contrário, sendo os valores armazenados do dicionário *g3c* como *1* e *0*, respectivamente.

```
obs = {}
Qui = {}
var = 0.0
gl = {}
for x in D.keys():
    obs[x] = 0
    for p in range(0,len(D[x]),6):
```

```
            obs[x] = obs[x] + dado[D[x][p],j,D[x][p+2],jj,D[x][p+4],jjj]
        var = var + (((obs[x]-Q[x])**2)/Q[x])
    Qui[j,jj,jjj] = var
    #print "Qui[", j , ",", jj, ",", jjj, "] =", Qui[j,jj,jjj]
    gl[j,jj,jjj] = int(len(D.keys())-1)
    #print "gl[", j, ",", jj, ",", jjj, "] =", gl[j,jj,jjj]
    # Teste do Qui-Quadrado - verifica se 3 colunas são ou não dependentes:
    #print "QQ[", gl[j,jj,jjj], "][2] =", float(QQ[gl[j,jj,jjj]][2])
    if (Qui[j,jj,jjj] >= float(QQ[gl[j,jj,jjj]][2])):
        g3c[j,jj,jjj] = 1
        print "As colunas", j, ",", jj, "e", jjj, "são dependentes"
    else:
        g3c[j,jj,jjj] = 0
        print "As colunas", j, ",", jj, "e", jjj, "são independentes"

#print "g3c =", g3c
```

Ao executar o presente programa, as triplas obtidos de colunas independentes são as seguintes:

colunas 3, 4 *e* 12, *colunas* 3, 8 *e* 12, *colunas* 3, 12 *e* 13, *colunas* 4, 12 *e* 16, *colunas* 5, 12 *e* 13, *colunas* 8, 11 *e* 13, *colunas* 8, 12 *e* 13, *colunas* 8, 12 *e* 16, *colunas* 8, 13 *e* 16, *colunas* 11, 13 *e* 16, e *colunas* 12, 13 *e* 16

As restantes *444* triplas de colunas, não mencionadas, são dependentes.

Da mesma forma que no programa anterior, quando se diz que a tripla de colunas 3, 4 *e* 12 é independente, em termos genéticos, pretende-se dizer que há ausência de associação entre os marcadores presentes nas colunas 3, 4 *e* 12 do ficheiro $YPortugal_Total2.csv$.

3.4 2colunas+3colunas_Contiguidade.py

Neste programa pretende-se testar a possibilidade de ter três colunas dependentes, sem que as mesmas, duas a duas, o sejam, no caso em que se agrupam apenas valores contíguos.

Começa-se, então, por chamar, $file_2c$ e $file_3c$, e posteriormente, executar os programas, já elaborados, e descritos anteriormente, *2colunas_Contiguidade.py* e *3colunas_Contiguidade.py*, os quais estudam a dependência e independência das colunas duas a duas e três a três, respectivamente, no caso em que se admite contiguidade.

```
file_2c = open("2colunas_Contiguidade.py")
exec file_2c
file_3c = open("3colunas_Contiguidade.py")
exec file_3c
```

Posteriormente, efectua-se o teste do *Qui-Quadrado*, para cada conjunto de três colunas, no qual, à semelhança do que foi feito nos programas acima mencionados, as segunda e terceira colunas são posteriores, respectivamente, às primeira e segunda colunas. Começa-se, portanto, por verificar se as mesmas são independentes, caso em que o valor de *g3c[j,jj,jjj]* é *0*. Em caso afirmativo, estuda-se a veracidade simultânea das três listas *g2c[j,jj]*, *g2c[j,jjj]* e *g2c[jj,jjj]* serem nulas, caso em que as colunas *j*, *jj* e *jjj* são independentes duas a duas. Em caso afirmativo das duas condições impostas, conclui-se a dependência das três colunas, mas independência das mesmas, duas a duas.

```
# Teste do Qui-Quadrado para verificar se 3 colunas podem ser dependentes, sem que as
mesmas colunas, 2 a 2, sejam dependentes.

for j in range(nc_nao,nc):
    if (j+1 < nc):
        for jj in range(j+1,nc):
            if (jj+1 < nc):
```

```
for jjj in range(jj+1,nc):
  if (g3c[j,jj,jjj] == 1):
    if ((g2c[j,jj] == 0) and (g2c[j,jjj] == 0) and (g2c[jj,jjj] == 0)):
      print "As colunas", j, ",", jj, "e", jjj, "são dependentes, mas indepen-
dentes 2 a 2."
```

Na execução deste programa, obtêem-se as seguintes colunas, dependentes três a três, mas independentes duas a duas:

colunas 2, 13 *e* 16 e *colunas* 4, 11 *e* 16

Ao dizer que as triplas de colunas mencionadas são dependentes três a três mas independentes duas a duas, em termos de genética, pretende-se dizer que há associação entre os marcadores representados pelas mesmas, no ficheiro $YPortugal_Total2.csv$, não havendo, porém, associação entre os marcadores, quando as colunas são consideradas duas a duas.

3.5 2colunas_Não Contiguidade.py

Neste programa pretende-se aplicar o teste do *Qui-Quadrado*, para determinar a dependência ou independência das colunas do ficheiro dado, duas a duas. À semelhança do que é efectuado no programa $2colunas_Contiguidade.py$, aplica-se o teste referido, tendo em conta apenas valores esperados maiores que *5*, pelo que se efectua o agrupamento dos diferentes valores que não obedecem à esta condição. No entanto, ao contrário do programa referido, à junção dos valores não é imposta nenhuma restrição, pelo que valores **não-contíguos** podem ser agrupados.

É de salientar que o presente programa é elaborado com a finalidade de constatar semelhanças e diferenças com o programa já mencionado, no que diz respeito aos resultados de dependência ou independência das colunas do ficheiro *YPortugal_Total2.csv*. Com esta comparação de resultados pretende-se analisar a influência do uso da condição de contiguidade.

É de notar a semelhança, nos passos efectuados, com o programa mencionado anteriormente, visto que, do anterior para este, apenas se retirou a condição que impunha a contiguidade. Por este motivo, apenas serão indicadas as diferenças ao longo do programa.

```
# Chamar o ficheiro Dados.py, que contém os dados necessários à execução do programa.
chamar = open("Dados.py")
exec chamar

# dado[l,j,ll,jj] indica o número de linhas observadas com o valor l na coluna j e o valor ll
na coluna jj.
# dadototal[l,j,ll,jj] indica o número de linhas observadas com o valor l na coluna j e o
valor ll na coluna jj, tendo
# em conta o número total de linhas do ficheiro.

dado = {}
dadototal = {}
for j in range(nc_nao,nc):
    if j+1 < nc:
        for jj in range(j+1,nc):
            for l in coluna[j]:
                for ll in coluna[jj]:
                    h = 0
                    for i in range( nl_nao , nl ):
                        if ((int (t[i][j]) == l) and (int (t[i][jj]) == ll)):
                            h = h + 1.0
```

```
            dado[l,j,ll,jj] = h
            #print "dado[", l, ",", j , ",", ll, ",", jj , "] =", dado[l,j,ll,jj]
            dadototal[l,j,ll,jj] = dado[l,j,ll,jj] / (nl-nl_nao)
            #print "dadototal[", l, ",", j , ",", ll, ",", jj , "] =", dadototal[l,j,ll,jj]


# probcoluna[l,j] representa a probabilidade de ter o valor l na coluna j.

probcoluna = {}
for j in range(nc_nao,nc):
   for l in coluna[j]:
      probcoluna[l,j] = obscoluna[l,j]/(nl-nl_nao)
      #print "probcoluna[", l, ",", j, "] =", probcoluna[l,j]


# D[i] é a lista que representa o i-ésimo par da forma [l,j,ll,jj], ié, valores l e ll nas colunas
j e jj, respectivamente.
# Q[i] é a lista que representa o número de linhas esperado com o par da forma [l,j,ll,jj].
# Listas D e Q, após a junção dos pequeninos entre si e junção dos pequeninos ao grande
mais próximo.
# Qui[j,jj] representa a estatística do teste Qui-Quadrado, para as colunas j e jj.
# gl[j,jj] dá-nos os graus de liberdade entre as colunas j e jj.

g2c = {}
for j in range(nc_nao,nc):
   if (j+1 < nc):
      for jj in range(j+1,nc):
         i = 0
         D = {}
         Q = {}
         for l in coluna[j]:
            for ll in coluna[jj]:
```

```
D[i] = [l,j,ll,jj]
Q[i] = probcoluna[l,j]*probcoluna[ll,jj]*(nl-nl_nao)
i = i + 1
```

Nos passos seguintes, procede-se a um agrupamento dos valores do domínio, com valor esperado menor que *5*, por ordem crescente.

Ao contrário do que foi efectuado no programa *2colunas_Contiguidade.py*, neste não se procede ao seguinte cálculo, que sendo igual a 1, impõe a contiguidade dos valores:

$$(D[z][p] - D[t][pp])^2 + (D[z][p+2] - D[t][pp+2])^2$$

uma vez que os valores agrupados não terão que ser, obrigatoriamente, contíguos. Assim sendo, no passo seguinte, apenas se impõe a condição $Q[t] < k - h$, comum ao programa acima referido.

```
w = 0
while (w == 0):
    w = 1
    for z in D.keys():
        ww = 0
        for p in range(0,len(D[z]),4):
            for k in range(2,8):
                for h in range(int(k/2),0,-1):
                    if (Q[z] < h):
                        for t in D.keys():
                            if (t != z):
                                if (Q[t] < k-h):
                                    w = 0
                                    D[z] = D[z] + D[t]
```

```
                    Q[z] = Q[z] + Q[t]
                    del D[t]
                    del Q[t]
                    ww = 1
                    break
        if (ww == 1):
            break
    #print "D =", D
    #print "Q =", Q
```

Nos passos seguintes, pretende-se agrupar os valores do domínio, com valor esperado menor que *5*, que não são passíveis de agrupar a outro elemento nas mesmas condições, a um valor com valor esperado maior que *5*, por ordem decrescente.

Pelo mesmo motivo que o referido no passo anterior, não se procede ao cálculo da condição de Contiguidade:

$$(D[z][p] - D[t][pp])^2 + (D[z][p+2] - D[t][pp+2])^2$$

Assim sendo, no passo seguinte, faz-se a mudança da variável z para o valor y, quando o valor z é *99*, ou quando $Q[y] < Q[z]$, não sendo necessário impor a contiguidade dos valores.

```
for k in range(5,0,-1):
    H = 0
    while (H == 0):
        H = 1
        for x in D.keys():
            ww = 0
            for p in range(0,len(D[x]),4):
```

```
            if ((Q[x] < k) and (Q[x] >= k-1)):
                z = 99
                for y in D.keys():
                    if (y != x):
                        if (z == 99):
                            z = y
                        else:
                            if (Q[y] < Q[z]):
                                z = y
                        D[x] = D[x] + D[y]
                        Q[x] = Q[x] + Q[y]
                        del D[y]
                        del Q[y]
                        H = 0
                        ww = 1
                        break
        if (ww == 1):
            break
#print "D =", D
#print "Q =", Q
```

Os passos seguintes são os mesmos do programa *2colunas_Contiguidade.py*, pelo que não serão explicados novamente, em virtude de se encontrarem pormenorizadamente descritos no referido.

```
for x in coluna[j]:
    for y in coluna[jj]:
        for z in D.keys():
```

```
            g = 0
            for w in range(0,len(D[z])-3,4):
               if ((x == D[z][w]) and (y == D[z][w+2])):
                  #print z+10,
                  g = 1
                  break
            if g == 1:
               break
      #print
   #print
   obs = {}
   Qui = {}
   var = 0.0
   gl = {}
   for x in D.keys():
      obs[x] = 0
      for p in range(0,len(D[x]),4):
         obs[x] = obs[x] + dado[D[x][p],j,D[x][p+2],jj]
      var = var + (((obs[x]-Q[x])**2)/Q[x])
   Qui[j,jj] = var
   #print "Qui[", j , ",", jj, "] =", Qui[j,jj]
   gl[j,jj] = int(len(D.keys())-1)
   #print "gl[", j, ",", jj, "] =", gl[j,jj]
   # Teste do Qui-Quadrado - verifica se 2 colunas são ou não dependentes:
   #print "QQ[", gl[j,jj], "][2] =", float(QQ[gl[j,jj]][2])
   if (Qui[j,jj] >= float(QQ[gl[j,jj]][2])):
      g2c[j,jj] = 1
      print "As colunas", j, "e", jj, "são dependentes"
   else:
```

```
g2c[j,jj] = 0
print "As colunas", j, "e", jj, "são independentes"


#print "g2c =", g2c
```

Analogamente aos programas anteriores, ao executar o programa descrito, os pares de colunas obtidos, independentes duas a duas, são os seguintes:

colunas 2 *e* 13, *colunas* 3 *e* 4, *colunas* 3 *e* 13, *colunas* 4 *e* 11, *colunas* 4 *e* 12, *colunas* 4 *e* 13, *colunas* 5 *e* 12, *colunas* 5 *e* 13, *colunas* 6 *e* 13, *colunas* 8 *e* 11, *colunas* 8 *e* 13, *colunas* 11 *e* 13, *colunas* 11 *e* 16, *colunas* 12 *e* 13, *colunas* 12 *e* 16, *colunas* 13 *e* 14, *colunas* 13 *e* 15, e *colunas* 13 *e* 16

Os restantes *87* pares de colunas, não mencionados, são dependentes.

Da mesma forma que no programa *2colunas_Contiguidade.py*, quando se diz que o par de colunas 2 *e* 13 é independente, em termos de genética, pretende-se dizer que há ausência de associação entre os marcadores presente nas colunas 2 *e* 13 do ficheiro *YPortugal_Total2.csv*. A única diferença entre o presente programa e o mencionado é a condição de contiguidade, que neste programa não foi imposta, provocando alterações, como se pode constactar, nos resultados obtidos. Os pares de colunas *2 e 16*, *4 e 16* e *8 e 12*, que eram independentes, quando se impunha contiguidade na junção de valores, deixaram de o ser quando essa condição deixou de ser imposta. Todavia, os pares de colunas *4 e 13* e *12 e 16*, que eram dependentes, passaram a ser independentes. Verifica-se, portanto, uma diminuição dos casos de independência, numa unidade, e um aumento dos casos de dependência, também numa unidade, quando a contiguidade dos valores não é considerada.

3.6 3colunas_Não Contiguidade.py

Neste programa aplica-se o teste do *Qui-Quadrado*, para determinar a dependência ou independência das colunas do ficheiro dado, três a três. Da mesma forma que no programa *3colunas_Contiguidade.py*, aplica-se o teste referido, apenas para valores com valor esperado maior que *5*, pelo que se agrupam os diferentes valores que não têm o valor esperado maior que *5*. Todavia, ao contrário do que se impôs no programa referido, neste não se impõe contiguidade à junção dos valores.

Note-se a semelhança deste programa com o programa descrito anteriormente, *2colunas_Não Contiguidade.py*, à excepção do número de colunas. O programa referido é aplicado a duas colunas, e este é aplicado a três, com a mesma finalidade.

Visto o presente programa ser semelhante aos dois referidos, os passos seguidos não vão ser pormenorizadamente descritos, nem exemplificados.

```
# Chamar o ficheiro Dados.py, que contém os dados necessários à execução do programa.
chamar = open("Dados.py")
exec chamar

# dado[j,l,jj,ll,jjj,lll] indica o número de linhas observadas com os valores l, ll e lll nas colunas j, jj e jjj, respectivamente.
# dadototal[j,l,jj,ll,jjj,lll] indica o número de linhas observadas com com os valores l, ll e lll nas colunas j, jj e jjj, respectivamente, tendo em conta o número total de linhas do ficheiro.

dado = {}
dadototal = {}
for j in range(nc_nao,nc):
    if j+1 < nc:
        for jj in range(j+1,nc):
            if jj+1 < nc:
```

```
            for jjj in range(jj+1,nc):
                for l in coluna[j]:
                    for ll in coluna[jj]:
                        for lll in coluna[jjj]:
                            h = 0
                            for i in range( nl_nao , nl ):
                                if ((int (t[i][j]) == l) and (int (t[i][jj]) == ll) and (int (t[i][jjj]) ==
lll)):
                                    h = h + 1.0
                            dado[l,j,ll,jj,lll,jjj] = h
                            #print "dado[", l, ",", j, ",", ll, ",", jj, ",", lll, ",", jjj, "] =",
dado[l,j,ll,jj,lll,jjj]
                            dadototal[l,j,ll,jj,lll,jjj] = dado[l,j,ll,jj,lll,jjj] / (nl-nl_nao)
                            #print "dadototal[", l, ",", j, ",", ll, ",", jj, ",", lll, ",", jjj, "] =",
dadototal[l,j,ll,jj,lll,jjj]


# probcoluna[l,j] representa a probabilidade de ter o valor l na coluna j.

probcoluna = {}
for j in range(nc_nao,nc):
    for l in coluna[j]:
        probcoluna[l,j] = obscoluna[l,j]/(nl-nl_nao)
        #print "probcoluna[", l, ",", j, "] =", probcoluna[l,j]


# D[i] é a lista que representa o i-ésimo par da forma [l,j,ll,jj,lll,jjj], ié, valores l e ll nas
colunas j e jj, resp.
# Q[i] é a lista que representa o número de linhas esperado com o par da forma [l,j,ll,jj,lll,jjj].
# Listas D e Q, após a junção dos pequeninos entre si e junção dos pequeninos ao grande
mais próximo.
# Qui[j,jj,jjj] representa a estatística do teste Qui-Quadrado, para as colunas j e jj.
```

```
# gl[j,jj,jjj] dá-nos os graus de liberdade entre as colunas j e jj.

g3c = {}
for j in range(nc_nao,nc):
    if (j+1 < nc):
        for jj in range(j+1,nc):
            if (jj+1 < nc):
                for jjj in range(jj+1,nc):
                    i = 0
                    D = {}
                    Q = {}
                    for l in coluna[j]:
                        for ll in coluna[jj]:
                            for lll in coluna[jjj]:
                                D[i] = [l,j,ll,jj,lll,jjj]
                                Q[i] = probcoluna[l,j]*probcoluna[ll,jj]*probcoluna[lll,jjj]*(nl-nl_nao)
                                i = i + 1
```

Nos próximos passos agrupam-se os valores do domínio, com valor esperado menor que *5*, por ordem crescente.

Neste programa não se calcula a condição de contiguidade de valores, caso em que a mesma é igual a 1, ao contrário do programa *3colunas_Contiguidade.py*. À semelhança do programa *2colunas_Não Contiguidade.py*, não se determina a condição referida:

$$(D[z][p] - D[t][pp])^2 + (D[z][p+2] - D[t][pp+2])^2 + (D[z][p+4] - D[t][pp+4])^2$$

e, no passo seguinte, apenas se impõe a condição $Q[t] < k - h$.

```
w = 0
while (w == 0):
    w = 1
    for z in D.keys():
        ww = 0
        for p in range(0,len(D[z]),6):
            for k in range(2,8):
                for h in range(int(k/2),0,-1):
                    if (Q[z] < h):
                        for t in D.keys():
                            if (t != z):
                                if (Q[t] < k-h):
                                    w = 0
                                    D[z] = D[z] + D[t]
                                    Q[z] = Q[z] + Q[t]
                                    del D[t]
                                    del Q[t]
                                    ww = 1
                                    break
        if (ww == 1):
            break
#print "D =", D
#print "Q =", Q
```

Nos passos seguintes, agrupam-se os valores do domínio, com valor esperado menor que *5*, a um valor com valor esperado maior que *5*, por ordem decrescente.

Da mesma forma que anteriormente, não se efectua o cálculo da condição de contiguidade:

$$(D[z][p] - D[t][pp])^2 + (D[z][p+2] - D[t][pp+2])^2 + (D[z][p+4] - D[t][pp+4])^2$$

No passo seguinte, não é portanto necessário impor a contiguidade dos valores e, tal como no programa *2colunas_Não Contiguidade.py* faz-se a mudança da variável z para o valor y, quando o valor z é *99*, ou quando $Q[y] < Q[z]$.

```
for k in range(5,0,-1):
    H = 0
    while (H == 0):
        H = 1
        for x in D.keys():
            ww = 0
            for p in range(0,len(D[x]),6):
                if ((Q[x] < k) and (Q[x] >= k-1)):
                    z = 99
                    for y in D.keys():
                        if (y != x):
                            if (z == 99):
                                z = y
                            else:
                                if (Q[y] < Q[z]):
                                    z = y
                            D[x] = D[x] + D[y]
                            Q[x] = Q[x] + Q[y]
                            del D[y]
                            del Q[y]
                            H = 0
                            ww = 1
```

```
                    break
            if (ww == 1):
                break
#print "D =", D
#print "Q =", Q
```

Os passos seguintes são os mesmos do programa *3colunas_Contiguidade.py*, e tal como no programa anterior, estes não serão descritos novamente, visto já se encontrarem pormenorizadamente descritos.

```
for x in coluna[j]:
    for y in coluna[jj]:
        for t in coluna[jjj]:
            for z in D.keys():
                g = 0
                for w in range(0,len(D[z])-5,6):
                    if ((x == D[z][w]) and (y == D[z][w+2]) and (t == D[z][w+4])):
                        #print z+100,
                        g = 1
                        break
                if g == 1:
                    break
        #print
    #print
obs = {}
Qui = {}
var = 0.0
gl =
```

```
            for x in D.keys():
                obs[x] = 0
                for p in range(0,len(D[x]),6):
                    obs[x] = obs[x] + dado[D[x][p],j,D[x][p+2],jj,D[x][p+4],jjj]
                var = var + (((obs[x]-Q[x])**2)/Q[x])
            Qui[j,jj,jjj] = var
            #print "Qui[", j , ",", jj, ",", jjj, "] =", Qui[j,jj,jjj]
            gl[j,jj,jjj] = int(len(D.keys())-1)
            #print "gl[", j, ",", jj, ",", jjj, "] =", gl[j,jj,jjj]
            # Teste do Qui-Quadrado - verifica se 3 colunas são ou não dependentes:
            #print "QQ[", gl[j,jj,jjj], "][2] =", float(QQ[gl[j,jj,jjj]][2])
            if (Qui[j,jj,jjj] ¿= float(QQ[gl[j,jj,jjj]][2])):
                g3c[j,jj,jjj] = 1
                print "As colunas", j, ",", jj, "e", jjj, "são dependentes"
            else:
                g3c[j,jj,jjj] = 0
                print "As colunas", j, ",", jj, "e", jjj, "são independentes"
#print "g3c =", g3c
```

Igualmente ao programa *3colunas_Contiguidade.py*, ao executar o presente programa, as triplas de colunas independentes, obtidas, são as seguintes:

colunas 3, 8 *e* 13, *colunas* 3, 12 *e* 13, *colunas* 4, 11 *e* 13, *colunas* 4, 12 *e* 16, *colunas* 5, 12 *e* 13, *colunas* 8, 11 *e* 13, *colunas* 8, 12 *e* 13, *colunas* 8, 12 *e* 16, *colunas* 8, 13 *e* 16, e *colunas* 12, 13 *e* 16

As restantes *445* triplas de colunas, não mencionadas, são dependentes.

Da mesma forma que no programa mencionado, quando se diz que a tripla de colunas 3, 8 *e* 13 é independente, em termos genéticos, pretende-se dizer que há ausência de associação entre os marcadores presentes nas colunas 3, 8 *e* 13 do ficheiro *YPortugal_Total2.csv*. A única diferença entre o presente programa e o mencionado é a condição de contiguidade, que neste programa não foi imposta, provocando alterações, como se pode constactar, nos resultados obtidos. Este facto já havia sido referido no programa anterior, em relação ao programa *2colunas_Contiguidade.py*, pelas mesmas razões.

Comparando os resultados obtidos nos dois programas, *3colunas_Contiguidade.py* e *3colunas_ Não Contiguidade.py*, constacta-se que as triplas de colunas *3, 4 e 12*, *3, 8 e 12* e *11, 13 e 16*, que eram independentes, quando se impunha contiguidade na junção de valores, deixaram de o ser quando essa condição deixou de ser imposta. Todavia, as triplas de colunas *3, 8 e 13* e *4, 11 e 13*, que eram dependentes, passaram a ser independentes. Analogamente aos programas *2colunas_Contiguidade.py* e *2colunas_ Não Contiguidade.py*, verifica-se uma diminuição dos casos de independência, numa unidade, e um aumento dos casos de dependência, também numa unidade, quando a contiguidade dos valores não é considerada.

3.7 2colunas+3colunas_Não Contiguidade.py

Este programa é semelhante ao programa *2colunas + 3colunas_Contiguidade.py*, testando-se a possibilidade de ter três colunas dependentes, sem que as mesmas, duas a duas, o sejam. No entanto, aqui não se impõe a contiguidade de valores.

À semelhança do programa referido, começa-se, por chamar, $file_2c$ e $file_3c$, e posteriormente, executar os programas, já elaborados, e descritos anteriormente, *2colunas_Não Contiguidade.py* e *3colunas_Não Contiguidade.py*, os quais estudam a dependência e independência das colunas duas a duas e três a três, respectivamente.

```
file_2c = open("2colunas_Não Contiguidade.py")
exec file_2c
file_3c = open("3colunas_Não Contiguidade.py")
exec file_3c
```

Posteriormente, efectua-se o teste do *Qui-Quadrado*, da forma já descrita no programa *2colunas + 3colunas_Contiguidade.py*.

```
# Teste do Qui-Quadrado para verificar se 3 colunas podem ser dependentes, sem que as
mesmas colunas, 2 a 2, sejam dependentes.

for j in range(nc_nao,nc):
   if (j+1 < nc):
      for jj in range(j+1,nc):
         if (jj+1 < nc):
            for jjj in range(jj+1,nc):
               if (g3c[j,jj,jjj] == 1):
                  if ((g2c[j,jj] == 0) and (g2c[j,jjj] == 0) and (g2c[jj,jjj] == 0)):
                     print "As colunas", j, ",", jj, "e", jjj, "são dependentes, mas indepen-
dentes 2 a 2."
```

Na execução do presente programa, obtêem-se as seguintes colunas, dependentes três a três, mas independentes duas a duas:

colunas 3, 4 *e* 13, *colunas* 4, 12 *e* 13 e *colunas* 11, 13 *e* 16

De forma análoga ao programa *2colunas+3colunas_Contiguidade.py*, relativamente às triplas de colunas mencionadas, quando se diz que as mesmas colunas são dependentes

três a três mas independentes duas a duas, em termos genéticos pretende-se dizer que há associação entre os marcadores representados pelas mesmas, no ficheiro *YPortugal_Total2.csv*, não havendo, porém, associação entre os marcadores, quandos os mesmos são considerados dois a dois.

Analogamente aos dois programas anteriores, a única diferença entre o presente programa e o programa mencionado anteriormente é a condição de contiguidade, que neste programa não foi imposta, o que justifica as alterações que se podem constactar nos resultados obtidos. Os casos de independência três a três, mas independência dois a dois, não se mantêm quando se assume contiguidade na junção de valores, e quando se deixa de considerar essa condição. Todavia, quando a condição de contiguidade não é imposta, o número de casos nas condições mencionadas aumenta numa unidade.

3.8 Ordenar por Probabilidades.py

O presente programa é idêntico ao programa *2colunas_Contiguidade.py*, aplicando-se o teste do *Qui-Quadrado*, para determinar a dependência ou independência das colunas do ficheiro dado, duas a duas, tendo em conta apenas valores esperados maiores que *5*, agrupando-se os diferentes valores **contíguos** que não obedecem à condição referida. A única diferença entre este programa e o referido reside na representação das colunas, por ordem decrescente de intervalos de probabilidade, *PP*, seguindo-se os graus de liberdade, *gl*, e o valor do *Qui-Quadrado*, *Qui*.

Chama-se e executa-se o ficheiro *Dado.py*, *chamar*, seguindo-se a determinação do número de linhas observadas com os valores *l* e *ll* nas colunas *j* e *jj*, respectivamente, *dado[l,j,ll,jj]*, e tendo em conta o número total de linhas do ficheiro, *dadototal[l,j,ll,jj]*, assim como a probabilidade de ter um valor *l* na coluna *j*, *probcoluna[l,j]*. Os passos referidos encontram-se pormenorizadamente descritos e exemplificados no programa atrás mencionado.

```
# Chamar o ficheiro Dados.py, que contém os dados necessários à execução do programa.

chamar = open("Dados.py")
exec chamar

# dado[l,j,ll,jj] indica o número de linhas observadas com o valor l na coluna j e o valor ll
na coluna jj.
# dadototal[l,j,ll,jj] indica o número de linhas observadas com o valor l na coluna j e o
valor ll na coluna jj, tendo em conta o número total de linhas do ficheiro.

dado = {}
dadototal = {}
for j in range(nc_nao,nc):
    if j+1 < nc:
        for jj in range(j+1,nc):
            for l in coluna[j]:
                for ll in coluna[jj]:
                    h = 0
                    for i in range( nl_nao , nl ):
                        if ((int (t[i][j]) == l) and (int (t[i][jj]) == ll)):
                            h = h + 1.0
                    dado[l,j,ll,jj] = h
                    #print "dado[", l, ",", j , ",", ll, ",", jj , "] =", dado[l,j,ll,jj]
                    dadototal[l,j,ll,jj] = dado[l,j,ll,jj] / (nl-nl_nao)
                    #print "dadototal[", l, ",", j , ",", ll, ",", jj , "] =", dadototal[l,j,ll,jj]


# probcoluna[l,j] representa a probabilidade de ter o valor l na coluna j.

probcoluna = {}
for j in range(nc_nao,nc):
    for l in coluna[j]:
        probcoluna[l,j] = obscoluna[l,j]/(nl-nl_nao)
```

```
#print "probcoluna[", l, ",", j, "] =", probcoluna[l,j]
```

Os próximos passos são semelhantes aos do programa acima referido, salientando-se apenas pequenas diferenças, as quais serão descritas de uma forma mais pormenorizada.

Relembre-se que no programa mencionado, primeiramente, se agrupam os valores, contíguos, dos domínios de D e Q com valor esperado menor que 5, por ordem crescente, e só depois se juntam os valores dos mesmos domínios, com valor esperado menor que 5, a um valor com valor esperado maior que 5, também por ordem decrescente.

```
# D[i] é a lista que representa o i-ésimo par da forma [l,j,ll,jj], ié, valores l e ll nas colunas
j e jj, respectivamente.
# Q[i] é a lista que representa o número de linhas esperado com o par da forma [l,j,ll,jj].
# cont[x,y] é a lista que indica se D[x] e D[y] são contiguos (= 1) ou não (= 0).
# Listas D e Q, após a junção dos pequeninos contíguos entre si e junção dos pequeninos
contíguos, ao grande mais próximo.
# Qui[j,jj] representa a estatística do teste Qui-Quadrado, para as colunas j e jj.
# gl[j,jj] dá-nos os graus de liberdade entre as colunas j e jj.

g2c = {}
gl_aux1 = 0
gl_aux2 = 100
Qui_aux1 = 0
Qui_aux2 = 100
Qui = {}
gl = {}
for j in range(nc_nao,nc):
    if (j+1 < nc):
        for jj in range(j+1,nc):
            i = 0
            D = {}
```

```
Q = {}
for l in coluna[j]:
    for ll in coluna[jj]:
        D[i] = [l,j,ll,jj]
        Q[i] = probcoluna[l,j]*probcoluna[ll,jj]*(nl-nl_nao)
        i = i + 1
cont =
w = 0
while (w == 0):
    w = 1
    for z in D.keys():
        C = 0
        ww = 0
        for p in range(0,len(D[z]),4):
            if (C == 1):
                break
            else:
                for k in range(2,8):
                    for h in range(int(k/2),0,-1):
                        if (Q[z] < h):
                            for t in D.keys():
                                if (t != z):
                                    for pp in range(0,len(D[t]),4):
                                        if (((D[z][p]-D[t][pp])**2)+((D[z][p+2]-D[t][pp+2])**2)==1):
                                            C = 1 # ié, D[z] contíguo com D[t]
                                            break
                                        else:
                                            C = 0
                                    cont[z,t] = C
```

```
                        if ((cont[z,t] == 1) and (Q[t] < k-h)):
                            w = 0
                            D[z] = D[z] + D[t]
                            Q[z] = Q[z] + Q[t]
                            del D[t]
                            del Q[t]
                            ww = 1
                            break
            if (ww == 1):
                break
#print "D =", D
#print "Q =", Q
for k in range(5,0,-1):
    H = 0
    while (H == 0):
        H = 1
        for x in D.keys():
            C = 0
            ww = 0
            for p in range(0,len(D[x]),4):
                if (C == 1):
                    break
                else:
                    if ((Q[x] < k) and (Q[x] >= k-1)):
                        z = 99
                        for y in D.keys():
                            if (y != x):
                                for pp in range(0,len(D[y]),4):
                                    if (((D[x][p]-D[y][pp])**2)+((D[x][p+2]-D[y][pp+2])**2) ==
```

```
1):
                                        C = 1 # ié, D[x] contíguo com D[y]
                                        break
                                    else:
                                        C = 0
                                cont[x,y] = C
                                if (cont[x,y] == 1):
                                    if (z == 99):
                                        z = y
                                    else:
                                        if (Q[y] < Q[z]):
                                            z = y
                                    D[x] = D[x] + D[y]
                                    Q[x] = Q[x] + Q[y]
                                    del D[y]
                                    del Q[y]
                                    H = 0
                                    ww = 1
                                    break
                        if (ww == 1):
                            break
    #print "D =", D
    #print "Q =", Q
    for x in coluna[j]:
        for y in coluna[jj]:
            for z in D.keys():
                g = 0
                for w in range(0,len(D[z])-3,4):
                    if ((x == D[z][w]) and (y == D[z][w+2])):
```

```
                    #print z+10,
                    g = 1
                    break
            if g == 1:
                break
        #print
    #print
```

Note-se que no inicio deste ciclo se consideraram quatro variáveis auxiliares, gl_aux1, gl_aux2, Qui_aux1 e Qui_aux2, com os respectivos valores, *0*, *100*, *0* e *100*.

Depois da junção dos valores dos domínios, nas condições pretendidas e referidas anteriormente, procede-se à determinação do valor do *Qui-Quadrado* para cada par de colunas, *j* e *jj*, *Qui[j,jj]*. Depois de determinado este valor, compara-se o valor obtido com os valores Qui_aux1 e Qui_aux2, e no caso do valor determinado ser maior que o primeiro ou menor que o segundo, o valor que se está a comparar é substituído por este. Segue-se este processo para todos os valores de *Qui-Quadrado* determinados, e no final, os valores Qui_aux1 e Qui_aux2 sao substituidos por Qui_max e Qui_min e representam, respectivamente, o maior e o menor valor do *Qui-Quadrado* calculado.

Segue-se um raciocínio análogo para os graus de liberdade. Depois de determinados os graus de liberdade para cada par de colunas *j* e *jj*, *gl[j,jj]*, o valor obtido é comparado com os valores gl_aux1 e gl_aux2, e à semelhança do caso anterior, no caso do valor obtido ser maior que o primeiro ou menor que o segundo, ele será substituído pelo valor comparado correspondente. Este processo é também seguido para todos os valores determinados, e no final os valores gl_aux1 e gl_aux2 sao substituídos por gl_max e gl_min e indicam, respectivamente, o maior e o menor valor de graus de liberdade calculado.

Neste ciclo é ainda realizado o teste do *Qui-Quadrado*, que verifica a dependência ou independência de cada duas colunas. O processo é o descrito no programa *2colunas_Contiguidade.py*.

```
obs = {}
#Qui = {}
var = 0.0
#gl = {}
for x in D.keys():
   obs[x] = 0
   for p in range(0,len(D[x]),4):
      obs[x] = obs[x] + dado[D[x][p],j,D[x][p+2],jj]
   var = var + (((obs[x]-Q[x])**2)/Q[x])
Qui[j,jj] = var
#print "Qui[", j , ",", jj, "] =", Qui[j,jj]
if (Qui[j,jj] < Qui_aux2):
   Qui_aux2 = Qui[j,jj]
if (Qui[j,jj] > Qui_aux1):
   Qui_aux1 = Qui[j,jj]
gl[j,jj] = int(len(D.keys())-1)
#print "gl[", j, ",", jj, "] =", gl[j,jj]
if (gl[j,jj] < gl_aux2):
   gl_aux2 = gl[j,jj]
if (gl[j,jj] > gl_aux1):
   gl_aux1= gl[j,jj]
# Teste do Qui-Quadrado - verifica se 2 colunas são ou não dependentes:
#print "QQ[", gl[j,jj], "][2] =", float(QQ[gl[j,jj]][2])
if (Qui[j,jj] >= float(QQ[gl[j,jj]][2])):
   g2c[j,jj] = 1
   print "As colunas", j, "e", jj, "são dependentes"
else:
   g2c[j,jj] = 0
   print "As colunas", j, "e", jj, "são independentes"
```

```
#print "g2c =", g2c
gl_min = gl_aux2
gl_max = gl_aux1
#print "gl_min", gl_min
#print "gl_max", gl_max
Qui_min = Qui_aux2
Qui_max = Qui_aux1
#print "Qui_min", Qui_min
#print "Qui_max", Qui_max
```

Posteriormente é definido o intervalo de probabilidade a que o valor do *Qui-Quadrado* de cada duas colunas, j e jj, pertence.

Note-se que, o ficheiro *Qui_Quadrado.csv* é composto por *6* colunas, sendo a primeira referente aos graus de liberdade, e as restantes às diferentes probabilidades, *0.1*, *0.05*, *0.025*, *0.01* e *0.005*. Nos passos que se seguem a primeira coluna não será considerada, uma vez que apenas interessa definir os intervalos de probabilidades associados ao valor do *Qui-Quadrado* de cada par de colunas. Assim, para cada par de colunas considera-se uma variável auxiliar n, inicialmente igual a *6*, e para cada uma das colunas do ficheiro referido, c, compara-se o valor do *Qui-Quadrado* determinado, com o do ficheiro, referente a essa coluna, e aos graus de liberdade correspondentes ao par de colunas em causa. Se o primeiro for menor que o segundo, então o valor n passa a tomar o valor c. Em seguida, consoante o valor de n, é atribuido um intervalo de probabilidade para cada par de colunas em causa, *PP[j,jj]*. Convém referir que para os valores *1*, *2*, *3*, *4* e *5*, de n, os intervalos de probabilidade associados são, respectivamente, *]0.1,1]*, *]0.05,0.1]*, *]0.025,0.05]*, *]0.01,0.025]*, *]0.005,0.01]* e *[0,0.005]*. Este processo repete-se, até que todos os pares de colunas tenham sido considerados.

Por exemplo,

$$Qui[2, 13] = 14.530008002 < \chi^2_{13,0.01} = 19.8119$$

pelo que o intervalo de probabilidade associado ao par de colunas 2 e 13 é $]0.1, 1]$.

```
# PP é a lista com os intervalos de probabilidade do teste do Qui-Quadrado, de cada 2 colunas.
# PP[j,jj] é o intervalo de probabilidade a que o valor Qui[j,jj] pertence.

PP = {}
for j in range(nc_nao,nc):
    if (j+1 < nc):
        for jj in range(j+1,nc):
            n = 6
            for c in range(1,6):
                if (Qui[j,jj] < float(QQ[gl[j,jj]][c])):
                    n = c
                    break
            if (n == 1):
                PP[j,jj] = "]0.1 , 1]"
            elif (n == 2):
                PP[j,jj] = "]0.05 , 0.1]"
            elif (n == 3):
                PP[j,jj] = "]0.025 , 0.05]"
            elif (n == 4):
                PP[j,jj] = "]0.01 , 0.025]"
            elif (n == 5):
                PP[j,jj] = "]0.005 , 0.01]"
            elif (n == 6):
                PP[j,jj] = "[0 , 0.005]"
            #print "PP[", j, ",", jj, "] =", PP[j,jj]
```

Para terminar, ordenam-se os dados por ordem decrescente de intervalos de probabillidade, *PP*, depois graus de liberdade, *gl*, e por fim valor do *Qui-Quadrado*, *Qui*.

Assim, começa-se por considerar uma variável k entre gl_min e gl_max, inclusive, e outra h entre Qui_min e Qui_max, inclusive, e para cada par de colunas, verifica-se se o intervalo de probabilidade é o pretendido. Em caso afirmativo, compara-se os graus de liberdade com o valor que a variável k está a tomar, e em caso de igualdade compara-se também o valor do Qui-Quadrado com o valor que a variável h está a tomar. No caso de se constatar também uma igualdade, ordenam-se os valores da seguinte forma:

colunas	*intervalo de probabilidade*	*graus de liberdade*	*valor do Qui-Quadrado*
[j,jj]	*PP[j,jj]*	*gl[j,jj]*	*Qui[j,jj]*

Note-se que se começa por trabalhar com o intervalo de probabilidade *]0.1,1]*, só depois se seguem os intervalos *]0.05,0.1]*, *]0.025,0.05]*, *]0.01,0.05]*, *]0.005,0.01]* e *[0,0.005]*, pela mesma ordem.

```
# Ordenar os dados, por PP, gl e Qui, por ordem decrescente.

print "colunas", "PP", "gl", "Qui"

for k in range(gl_max,gl_min-1,-1):
    for h in range(int(Qui_max),int(Qui_min-1),-1):
        for j in range(nc_nao,nc):
            if (j+1 < nc):
                for jj in range(j+1,nc):
                    if (PP[j,jj] == "]0.1 , 1]"):
                        if (gl[j,jj] == k):
                            if (int(Qui[j,jj]) == h):
                                print [j,jj], "]0.1 , 1]", k, Qui[j,jj]
```

```
for k in range(gl_max,gl_min-1,-1):
    for h in range(int(Qui_max),int(Qui_min-1),-1):
        for j in range(nc_nao,nc):
            if (j+1 < nc):
                for jj in range(j+1,nc):
                    if (PP[j,jj] == "]0.05 , 0.1]"):
                        if (gl[j,jj] == k):
                            if (int(Qui[j,jj]) == h):
                                print [j,jj], "]0.05 , 0.1]", k, Qui[j,jj]


for k in range(gl_max,gl_min-1,-1):
    for h in range(int(Qui_max),int(Qui_min-1),-1):
        for j in range(nc_nao,nc):
            if (j+1 < nc):
                for jj in range(j+1,nc):
                    if (PP[j,jj] == "]0.025 , 0.05]"):
                        if (gl[j,jj] == k):
                            if (int(Qui[j,jj]) == h):
                                print [j,jj], "]0.025 , 0.05]", k, Qui[j,jj]


for k in range(gl_max,gl_min-1,-1):
    for h in range(int(Qui_max),int(Qui_min-1),-1):
        for j in range(nc_nao,nc):
            if (j+1 < nc):
                for jj in range(j+1,nc):
                    if (PP[j,jj] == "]0.01 , 0.025]"):
                        if (gl[j,jj] == k):
                            if (int(Qui[j,jj]) == h):
                                print [j,jj], "]0.01 , 0.025]", k, Qui[j,jj]
```

```
for k in range(gl_max,gl_min-1,-1):
    for h in range(int(Qui_max),int(Qui_min-1),-1):
        for j in range(nc_nao,nc):
            if (j+1 < nc):
                for jj in range(j+1,nc):
                    if (PP[j,jj] == "]0.005 , 0.01]"):
                        if (gl[j,jj] == k):
                            if (int(Qui[j,jj]) == h):
                                print [j,jj], "]0.005 , 0.01]", k, Qui[j,jj]


for k in range(gl_max,gl_min-1,-1):
    for h in range(int(Qui_max),int(Qui_min-1),-1):
        for j in range(nc_nao,nc):
            if (j+1 < nc):
                for jj in range(j+1,nc):
                    if (PP[j,jj] == "[0 , 0.005]"):
                        if (gl[j,jj] == k):
                            if (int(Qui[j,jj]) == h):
                                print [j,jj], "[0 , 0.005]", k, Qui[j,jj]
```

Relembre-se que o presente programa é semelhante ao programa *2colunas_Contiguidade.py*, como já foi referido no início deste secção, e que a única diferença entre os mesmos reside na representação das colunas, por ordem decrescente de intervalos de probabilidade, *PP*, seguindo-se os graus de liberdade, *gl*, e o valor do *Qui-Quadrado*, *Qui*, que consta neste programa, e não no referido. Por este motivo, os resultados obtidos, relativamente aos casos de dependência e independência são exactamente os mesmos. No que refere a casos de independência, estes são citados, novamente, em seguida:

colunas 2 *e* 13, *colunas* 2 *e* 16, *colunas* 3 *e* 4, *colunas* 3 *e* 13, *colunas* 4 *e* 11, *colunas* 4 *e* 12, *colunas* 4 *e* 16, *colunas* 5 *e* 12, *colunas* 5 *e* 13, *colunas* 6 *e* 13, *colunas* 8 *e* 11, *colunas* 8 *e* 12, *colunas* 8 *e* 13, *colunas* 11 *e* 13, *colunas* 11 *e* 16, *colunas* 12 *e* 13, *colunas* 13 *e* 14, *colunas* 13 *e* 15, e *colunas* 13 *e* 16

Os restantes *86* pares de colunas, não mencionados, dizem respeito às colunas dependentes duas a duas.

Mais uma vez se recorda que, quando se diz que o par de colunas 2 *e* 13 é independente, em termos genéticos este facto significa que há ausência de associação entre os marcadores presentes nas colunas 2 *e* 13 do ficheiro $YPortugal_Total2.csv$.

Relativamente à representação referida, apresenta-se em seguida um pequeno extracto da mesma, obtida aquando da execução do programa:

colunas	*PP*	*gl*	*Qui*
[11, 16]	]0.1, 1]	16	19.3649812893
[13, 15]	]0.1, 1]	16	16.411952724
[4, 11]	]0.1, 1]	16	14.2555809742
[4, 12]	]0.1, 1]	15	16.0382745489
[5, 13]	]0.1, 1]	14	16.188294668
[12, 13]	]0.1, 1]	14	5.29705263781
[2, 13]	]0.1, 1]	13	14.530008002
[8, 12]	]0.1, 1]	13	11.3576420217
[3, 4]	]0.1, 1]	13	9.86491561162

3.9 Soma de n colunas.py

Neste programa pretende-se testar a dependência e independência de colunas, cuja soma de valores nas mesmas é imposta. Os passos seguidos para a execução do programa encontram-se seguidamente justificados.

Analogamente aos programas anteriores, começa-se por chamar e, posteriormente, executar o ficheiro *Dados.py*, com os dados essenciais à execução do programa, *chamar*.

```
# Chamar o ficheiro Dados.py, que contém os dados necessários à execução do programa.
chamar = open("Dados.py")
exec chamar
```

Posteriormente, insere-se o número de colunas que se pretende somar, de entre as colunas consideradas, *ncs*. Assim, poderão ser elaborados diferentes grupos de colunas, tendo cada um dos grupos *ncs* colunas para somar.

Por exemplo,

$$ncs = 3$$

quer dizer que serão consideradas 3 colunas na realização de cada um dos passos.

```
#ncs indica o número de colunas a somar
ncs = int(raw_input("insira o número de colunas a somar:\n"))
```

Em seguida é elaborada uma função, *suc(w)*, que nos dará o sucessor de uma lista w.

Convém salientar que w é uma lista formada por *0s* e *1s*, na qual o valor *1* indica que a coluna está a ser considerada para a soma e *0* caso contrário. Um exemplo de uma lista w, com $ncs = 3$, será:

$$w = [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0]$$

onde as colunas consideradas para a soma, neste caso, são a *8*, a *9* e a *10*. Note-se que as colunas *0* e *1* não são consideradas, como se poderá ver com a anterior determinação de *nc_nao*.

Para determinar a lista sucessora da lista w, verifica-se, do fim para o início da lista, se se encontra algum par da forma *0,1*. Caso seja encontrado, troca-se a ordem do mesmo, passando a ter o par da forma *1,0*. Posteriormente, as posições anteriores a este par permanecem inalteradas, e nas posições posteriores, contam-se aquelas em que aparece *1*, variável r, que inicialmente toma o valor nulo, e vai subindo um valor à medida que vai surgindo uma posição com *1* na lista. Seguidamente, colocam-se os r *1s* determinados nas posições finais da lista. Depois de distribuidos todos os *1s*, colocam-se *0s* nas restantes posições entre o par reordenado e o último *1* colocado. No exemplo anterior, *suc(w)* seria:

$$suc(w) = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1]$$

```
#suc(w) é a função que nos dá o sucessor de w

def suc( w ):
    for j in range(len(w)):
        if ((w[len(w)-j-1] == 1) and (w[len(w)-j-2] == 0)):
            w[len(w)-j-2] = 1
            w[len(w)-j-1] = 0
            r = 0
            for k in range(len(w)-j,len(w)):
                if (w[k] == 1):
                    r = r+1
            for kk in range(len(w)-r,len(w)):
                w[kk] = 1
            for kkk in range(len(w)-j,len(w)-r):
                w[kkk] = 0
```

```
        break
    return w
```

Para executar o presente programa, começa-se por definir a lista de onde se inicia o processo, *pali*, a qual tem *ncs 1s* no final da lista, sendo as restantes posições nulas. No caso de $ncs = 3$ a lista inicial seria:

$$pali = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1]$$

```
# pali é uma lista que representa a palavra inicial.
pali = []
for j in range(nc_nao,nc-ncs):
    pali = pali + [0]
for j in range(nc-ncs,nc):
    pali = pali + [1]
#print "pali =", pali
```

Seguidamente é elaborada outra função, *inv(w)*, que nos dará o inverso de uma lista w.

É de notar que a lista inversa de uma lista w, *inv(w)*, denotada na definição da função por v, consiste na colocação das posições da lista w por ondem inversa. No caso da lista w seguinte:

$$w = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0]$$

a sua inversa seria:

$$inv(w) = [0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]$$

Note-se ainda que, da mesma forma que foi necessário definir uma lista para iniciar o processo, também é necessário definir uma lista para terminá-lo, *palf*, a qual consiste na lista inversa de *pali*. No caso de $ncs = 3$ a lista final seria:

$$palf = [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
#inv(w) é a função que nos dá o inverso de w.
# palf é a lista que representa a palavra inversa de pali.

def inv( w ):
    v = []
    for i in range(len(w)):
        v = v + [w[len(w)-i-1]]
    return v
palf = inv( pali )
#print "palf =", palf
```

Em seguida encontra-se outra função, que consiste na conversão de uma lista, L, numa palavra, *est(L)*.

Partindo da palavra vazia, '', vai-se juntando as diferentes posições da lista considerada, neste caso L, pela ordem que as mesmas aparecem na lista. No caso da lista L ser da forma:

$$L = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0]$$

a sua transformação numa palavra seria:

$$est(L) = 000000001100100$$

```
# est(L) é a função que transforma uma lista numa palavra
def est( L ):
    u = ''
    for i in range(len(L)):
        u = u + str(L[i])
    return u
```

De seguida, determina-se o número de linhas observadas no ficheiro que têm um determinado valor para a soma, *soma*, das *ncs* colunas consideradas, isto é, com posições igual a *1* na lista *w*. O valor encontrado será armazenado na lista *dado[est(w),soma]*.

Começa-se por considerar a lista *w*, como sendo a lista inicial, *pali*, e inicia-se um ciclo que, apenas, será interrompido quando *w* passar a ser a lista final, *palf*.

Dentro do ciclo referido, tendo em conta apenas as posições *1* de *w*, isto é, as colunas que serão consideradas na soma dos respectivos valores, começa-se por somar os valores mais pequenos dessas colunas, *somainicial* (inicialmente nula), assim como os valores maiores das mesmas, *somafinal* (inicialmente também nula). Posteriormente, para qualquer valor de *soma* entre *somainicial* e *somafinal*, inclusive, considera-se uma variável, à priori nula, *h*, e para cada linha *i*, somam-se os valores que se encontram nas colunas indicadas por *1* em *w*, armazenando o resultado na variável *sl*, inicialmente nula, e que seguidamente se passará a chamar *somalinha[i]*. Em seguida, verifica-se se o valor obtido é igual ao valor *soma* considerado anteriormente. Em caso afirmativo, soma-se *1* à variável *h*. Depois de percorridas todas as linhas, a variável *h* denotar-se-á *dado[est(w),soma]* e será armazenada no dicionário *dado*.

O dicionário *dadototal* armazena as listas da forma *dadototal[est(w),soma]*, que indicam o número de linhas observadas no ficheiro original com um determinado valor para a soma, *soma*, das *ncs* colunas representadas por *1* na lista *w*, tendo em conta o número

total de linhas do ficheiro.

Para terminar, verifica-se se a lista w, que está a ser usada na execução do ciclo, é igual a *palf*. Em caso afirmativo, o ciclo é imediatamente interrompido, como já foi referido anteriormente, caso contrário, determina-se o sucessor de w, que se denota, inicialmente, por ww, substituindo, em seguida, o w anterior, e o ciclo volta a repetir-se.

```
# dado[est(w),soma] indica o número de linhas observadas cuja soma das colunas indicadas por 1 em w é soma.
#dadototal[est(w),soma] indica o número de linhas observadas cuja soma das colunas indicadas por 1 em w é soma, tendo em conta o número total de linhas do ficheiro.

dado = {}
dadototal = {}
somalinha = {}
w = [] + pali
#print "w =",w
while (0 == 0):
    somainicial = 0
    somafinal = 0
    for j in range(len(w)):
        if w[j] == 1:
            somainicial = somainicial + coluna[j+2][0]
            somafinal = somafinal + coluna[j+2][valores[j+2]-1]
    #print "somainicial", somainicial
    #print "somafinal", somafinal
    for soma in range (somainicial,somafinal+1):
        h = 0
        for i in range(nl_nao,nl):
            sl = 0
```

```
        for j in range(len(w)):
            if w[j] == 1:
                sl = sl + int(t[i][j+2])
        somalinha[i] = sl
        #print "somalinha[", i, "] =", somalinha[i]
        if (somalinha[i] == soma):
            h = h + 1
    dado[est (w ),soma] = h
    #print "dado[", est ( w ), ",", soma, "] =", dado[est( w ),soma]
    dadototal[est( w ),soma] = dado[est( w ),soma] / (nl-nl_nao)
    #print "dadototal["est( w ), ",", soma, "] =", dadototal[est( w ),soma]
if w == palf:
    break
else:
    ww = suc(w)
    w = ww
```

Por exemplo,

$$dado[000000001100100, 31] = 19.0$$

$$dadototal[000000001100100, 31] = 0.0289193302892$$

representam, respectivamente, o número de linhas observadas cuja soma dos valores nas colunas 10, 11 e 14 é 31, e o número de linhas observadas, nas mesmas condições, mas tendo em consideração o número total de linhas do ficheiro em estudo.

Seguidamente, para cada coluna considera-se um valor, l, entre *lmin* e *lmax*, inclusive, e do modo análogo ao anteriormente descrito nos programas precedentes, calcula-se a probabilidade de ter esse valor l na coluna j que está a ser considerada, *probcoluna[l,j]*.

```
# probcoluna[l,j] representa a probabilidade de ter o valor l na coluna j.

probcoluna = {}
for j in range(nc_nao,nc):
    for l in range(lmin,lmax+1):
        probcoluna[l,j] = obscoluna[l,j]/(nl-nl_nao)
        #print "probcoluna[", l, ",", j, "] =", probcoluna[l,j]
```

Posteriormente, é elaborada outra função, *men(w,k)*, que consiste na distribuição de um valor k, da menor forma possível, pelas posições *1*de w.

Para cada posição da lista w verifica-se se a mesma toma o valor *0* ou *1*. No caso da posições *0*, acrecenta-se à lista u, inicialmente nula, o valor *0*. Caso contrário, a respectiva posição em u toma o menor valor possivel na coluna correspondente. À medida que se vai atribuindo o menor valor da coluna à respectiva posição com 1 em w, o valor k, denominado por kk, desde inicio, é actualizado, passando a ser o valor kk anterior sem o valor que se atribui à posição em causa.

Depois de se ter atribuído um valor a cada posição da lista w, formando a lista u, verifica-se se o valor kk, actualizado, não é negativo, visto não fazer sentido que o seja. No caso desta condição se verificar, do final para o inicio da lista u, nas posições *1* de w, vai-se certificando se à correspondente posição em u se pode acrescentar o valor kk restante, sem o valor obtido exceder o máximo valor referente à coluna correspondente. Caso seja possível, então efectua-se o cálculo referido. Caso contrário, altera-se o valor dessa posição para o valor máximo possível na coluna correspondente, e volta-se a actualizar o valor kk, subtraindo ao valor anterior o valor que se acrescentou na posição considerada. Os passos referidos são efectuados sucessivamente, até se esgotar o valor kk.

No caso da lista w ser da forma:

$$w = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0]$$

e se se pretender distribuir um valor igual a *39* pelas colunas *10*, *11* e *14*, colunas correspondentes às posições *1* de w, a lista obtida será:

$$men(w, 39) = [0, 0, 0, 0, 0, 0, 0, 0, 9, 15, 0, 0, 15, 0, 0]$$

```
# men(w,k) define a menor distribuição de k elementos pelas colunas indicadas por 1 em
w.

def men(w,k):
    u = []
    kk = k
    for j in range(len(w)):
        if (w[j] == 0):
            u = u + [0]
        else:
            u = u + [coluna[nc+j-len(w)][0]]
            kk = kk - coluna[nc+j-len(w)][0]
        if (kk < 0):
            return []
        else:
            for i in range(len(w)):
                if (w[len(w)-i-1] == 1):
                    vv = coluna[nc-1-i][valores[nc-1-i]-1]
                    if (kk <= vv - u[len(w)-i-1]):
                        u[len(w)-i-1] = u[len(w)-i-1] + kk
                        break
                    else:
                        kk = kk - vv + u[len(w)-i-1]
                        u[len(w)-i-1] = vv
            return u
```

A seguinte função, *seg(u)*, permite obter uma lista sucessora da lista u, tendo em conta a soma das posições em u, representadas pelas posições *1* de w.

Tendo em conta a lista u que está a ser considerada, começa-se por definir uma lista w. Assim, as posições *0* de u definirão as posições *0* de w e as posições não nulas da lista u definirão as posições *1* de w. Seguidamente, apenas se trabalhará com as colunas correspondentes às posições representadas por *1* em w.

Seguindo um raciocínio análogo ao anterior, trabalha-se do final para o início da lista w, apenas nas posições *1*, procurando a primeira posição, em u, com um valor maior que o menor valor possível na coluna correspondente. Uma vez encontrado o valor, continua-se a percorrer a lista u, no mesmo sentido, apenas nas posições *1* de w como anteriormente, a partir da posição correspondente a esse valor até encontrar um valor menor que o maior valor possível na coluna correspondente. Uma vez encontrado, somam-se os valores posteriores a essa posição, *aux1*, e os menores valores possíveis nas posições posteriores referentes às posições *1* em w, *aux2*. Posteriormente, somo *1* à posição em causa, subtraindo então *1* à soma das posições posteriores, denotando este valor por *kl*. Seguidamente, distribui-se o valor *kl* pelas posições posteriores de u, representadas por *1* em w, usando a função definida anteriormente. As posições precedentes permanecem inalteradas. Para finalizar, interrompe-se o ciclo, uma vez que para obter a lista *seg(u)* basta percorrer o ciclo, na sua totalidade, uma única vez.

No caso da lista u em causa ser:

$$u = [0, 0, 0, 0, 0, 0, 0, 0, 9, 15, 0, 0, 15, 0, 0]$$

a lista sucessora da lista u, tendo em conta a soma das posições não nulas de u, será:

$$seg(u) = [0, 0, 0, 0, 0, 0, 0, 0, 10, 14, 0, 0, 15, 0, 0]$$

```
# seg(u) é a função que nos dá o sucessor de u, com soma k, nas colunas indicadas por 1 em w.

def seg(u):
    w = []
    for i in range(len(u)):
        if (u[i] == 0):
            w = w + [0]
        else:
            w = w + [1]
    xx = 0
    for j in range(len(u)):
        if (w[len(u)-j-1] == 1):
            if (u[len(u)-j-1] > coluna[len(u)-j+1][0]):
                xxx = 0
                for j2 in range(j+1, len(u)):
                    if (w[len(u)-j2-1] == 1):
                        if (u[len(u)-j2-1] < coluna[len(u)-j2+1][valores[len(u)-j2+1]-1]):
                            aux1 = 0
                            aux2 = 0
                            for jj in range(len(u)-j2,len(u)):
                                aux1 = aux1 + u[jj]
                                if u[jj] != 0:
                                    aux2 = aux2 + coluna[jj+2][0]
                            u[len(u)-j-1] = u[len(u)-j-1] + 1
                            ww = w[len(u)-j : len(u)]
                            kl = aux1 - 1
                            uu = men(ww,kl)
                            for jjj in range(len(uu)):
                                u[jjj+len(u)-j] = uu[jjj]
```

```
                    xx = 1
                    xxx = 1
                break
        if xxx == 1:
            break
if xx == 1:
    return u
else:
    return []
```

De seguida, determinam-se o número de linhas esperadas no ficheiro com um determinado valor para a soma, *soma*, das *ncs* colunas consideradas, isto é, das colunas com posição *1* na lista *w*. O valor determinado será armazenado na lista *esp[est(w),soma]*.

Começa-se por considerar a lista *pw*, como sendo a lista inicial, *pali*, e inicia-se um ciclo que, à semelhança do que foi efectuado na determinação do número de linhas observadas nas mesmas condições, apenas será interrompido quando *pw* passar a ser a lista final, *palf*.

Dentro do ciclo referido, tendo em conta apenas as posições *1* de *pw*, da mesma forma que anteriormente, começa-se por somar os valores mais pequenos dessas colunas, *somainicial* (inicialmente nula), assim como os valores maiores das mesmas, *somafinal* (inicialmente também nula). Posteriormente, para qualquer valor de *soma* entre *somainicial* e *somafinal*, inclusive, distribuo o valor pelas posições *1* de *pw*, da menor forma possível, usando a função *men(pw,soma)*, *u*, e considero uma variável *p*, inicialmente nula. Em seguida, inicia-se outro ciclo, dentro do anterior, o qual será interrompido quando a lista *u* for a lista nula, *[]*. No caso desta condição não ser satisfeita, considera-se uma variável *h*, inicialmente igual a *1.0*, e tendo em conta apenas as posições não nulas da lista *u*, multiplicam-se as probabilidades de ter cada um dos valores das posições de *u*, nas colunas correspondentes, por *h*, denotando o resultado obtido, também, por *h*. Depois deste cálculo, soma-se o valor *h* determinado ao valor *p*, denotando-o, também, por *p*. De seguida, determina-se a lista sucessora da lista *u*, com a mesma soma das mesmas posições não nulas, denotando-a, posteriormente, por *u*, e os procedimentos anteriores repetem-se,

como já foi referido, enquanto a lista *u* for diferente da lista nula.

Depois de terminado este ciclo, multiplica-se o valor *p* obtido pelo número total de linhas consideradas do ficheiro, armazenando esta informação na lista *esp[est(pw),soma].*

Para terminar, verifica-se se a lista *pw*, que está a ser usada na execução do ciclo, coincide com a lista final, *palf.* Em caso afirmativo, o ciclo é imediatamente interrompido, como já foi referido anteriormente. Caso contrário, determina-se o sucessor de *pw*, que se denota, inicialmente, por *z*, substituindo, em seguida, o *pw* anterior, e o ciclo volta a repetir-se.

Note-se que à medida que se vai calculando o produto das probabilidades, vai-se somando o valor determinado à variável *pt*, inicialmente nula. O objectivo deste cálculo é certificar que, para cada *pw* que se considera, no final a variável *pt* toma o valor *1.*

```
# esp[esp(w),soma] indica o número esperado de linhas cuja soma das colunas indicadas por 1 em w é soma, admitindo independência entre as colunas.
esp = {}
pw = [] + pali
while (0 == 0):
    somainicial = 0
    somafinal = 0
    for j in range(len(pw)):
        if (pw[j] == 1):
            somainicial = somainicial + coluna[j+2][0]
            somafinal = somafinal + coluna[j+2][valores[j+2]-1]
    #print "somainicial", somainicial
    #print "somafinal", somafinal
    pt = 0.0
    for soma in range(somainicial,somafinal+1):
        #print "soma =", soma
        u = men(pw,soma)
```

```
        #print "u =", u
        p = 0.0
        while (u != []):
            h = 1.0
            for jj in range(len(u)):
                if (u[jj] != 0):
                    h = h*probcoluna[u[jj],jj+2]
            p = p + h
            pt = pt + h
            useg = seg(u)
            u = useg
            #print "u =", u
        esp[est(pw),soma] = p*(nl-nl_nao)
        #print "esp[", est(pw), ",", soma, "] =", esp[est(pw),soma]
    if pw == palf:
        break
    else:
        z = suc(pw)
        pw = z
        #print "pw =", pw
        #print "pt =", pt
```

Por exemplo,

$$esp[000000001100100, 31] = 13.7095255636$$

representam o número de linhas esperadas cuja soma dos valores nas colunas 10, 11 e 14 é 31.

No próximo passo calcula-se o o valor do *Qui-Quadrado*, para cada lista w.

Seguindo um raciocínio análogo aos anteriores, começa-se por considerar a lista w, como sendo a lista inicial, *pali*, e inicia-se um ciclo que apenas será interrompido quando a lista w passar a ser a lista final, *palf*. Assim, dentro do ciclo referido, tendo em conta apenas as posições *1* de w, começa-se por somar os valores mais pequenos dessas colunas, *somainicial* (inicialmente nula), assim como os valores maiores das mesmas, *somafinal* (inicialmente também nula). Posteriormente, considera-se uma variável Q, à priori nula, e, para qualquer valor de *soma* entre *somainicial* e *somafinal*, inclusive, adiciona-se à variável Q a parcela:

$$\frac{(dado[est(w), soma] - esp[est(w), soma])^2}{esp[est(w), soma]}$$

Depois de efectuado este cálculo para todos os valores de *soma* possíveis, denota-se a variável Q por *Qui[est(w)]*.

Seguidamente, como nos casos anteriores, verifica-se se a lista w, que está a ser usada na execução do ciclo, é igual a *palf*. Em caso afirmativo, o ciclo é imediatamente interrompido. Caso contrário, determina-se o sucessor de w, que se denota, inicialmente, por *ww*, substituindo, em seguida, o w anterior, e o ciclo volta a repetir-se.

```
# Qui[est(w)] representa a estatística do teste Qui-Quadrado, para as colunas assinaladas
com 1 em w.

Qui = {}
w = [] + pali
#print"w =", w
while (0 == 0):
    somainicial = 0
    somafinal = 0
    for j in range(len(w)):
        if (w[j] == 1):
            somainicial = somainicial + coluna[j+2][0]
            somafinal = somafinal + coluna[j+2][valores[j+2]-1]
```

```
#print "somainicial", somainicial
#print "somafinal", somafinal
Q = 0.0
for soma in range (somainicial,somafinal+1):
   Q = Q + (((dado[est(w),soma]-esp[est(w),soma])**2)/esp[est(w),soma])
Qui[est(w)] = Q
#print "Qui[", est(w), "]=", Qui[est(w)]
if w == palf:
   break
else:
   ww = suc(w)
   w = ww
```

No caso de se ter:

$$Qui[000000001100100] = 129.166659618$$

significa que o valor do *Qui-Quadrado* associado às colunas 10, 11 e 14 é 129.166659618.

No próximo passo começa-se por considerar a lista w igual à lista inicial, *pali*, e inicia-se um ciclo que, de forma análoga aos casos anteriores, será interrompido quando a lista w coincidir com a lista final, *palf*. No primeiro passo do ciclo determina-se os graus de liberdade correspondentes ao w em causa, o qual corresponderá ao valor *ncs*, isto é, o número de colunas que se pretende somar, menos uma unidade.

Posteriormente, é realizado o teste do *Qui-Quadrado*, o qual consiste na determinação da dependência ou independência das colunas assinaladas por *1* em w, comparando o valor do *Qui-Quadrado* calculado com o tabelado. No caso do primeiro ser maior ou igual que o segundo, as colunas em causa dizem-se dependentes. Caso contrário as colunas dizem-se independentes.

Enquanto w for diferente de *palf*, determina-se o sucessor de w, ww, denotando-o em seguida por w, e o ciclo repete-se. Caso contrário, é imediatamente interrompido.

```
# Teste do Qui-Quadrado - verifica se 2 colunas são ou não dependentes.
# gl[est(w)] dá-nos os graus de liberdade entre as colunas assinaladas com 1 em w.
# Qui[est(w)] representa a estatística do teste Qui-Quadrado, para as colunas assinaladas
com 1 em w.

gl = {}
w = [] + pali
#print"w =", w
while (0 == 0):
    gl[est(w)] = ncs-1
    #print "QQ[", gl[est(w)], "][", 2,"] =", float(QQ[gl[est(w)]][2])
    if (Qui[est(w)] >= float(QQ[gl[est(w)]][2])):
        print "As colunas assinaladas com 1 em", est(w), "são dependentes"
    else:
        print "As colunas assinaladas com 1 em", est(w), "são independentes"
    if w == palf:
        break
    else:
        ww = suc(w)
        w = ww
```

No caso anteriormente exemplificado (000000001100100), tem-se:

$$gl = \textit{número de colunas consideradas} - 1 = 3 - 1 = 2$$

e o valor do *Qui-Quadrado tabelado* é:

$$\chi^2_{2,0.05} = 5.9915$$

Comparando com o valor anterior, verifica-se que este é menor, concluindo-se que as colunas 10, 11 e 14 são dependentes.

Ao executar o programa descrito, para $ncs = 2$, ou seja, somando apenas duas colunas, no que diz respeito a casos de independência, os resultados obtidos são os seguintes:

000000000011000 e 010000100000000

Os restantes *103* casos possíveis, não mencionados, dizem respeito a casos de dependência.

É de notar que, quando se diz que 000000000011000 é um caso de independência, se pretende dizer que as colunas 12 *e* 13 são independentes (relembre-se que no *excel* se inicia a contagem das colunas em 0, e como no ficheiro *YPortugal_Total2.csv* as colunas não consideradas são as duas primeiras, $nc_nao = 1$, *coluna 0* e *coluna 1*, no presente programa inicia-se a contagem das colunas em 2). Em termos de genética, este resultado significa que há ausência de associação entre os marcadores representados pelas colunas 12 *e* 13 do ficheiro *YPortugal_Total2.csv*.

Comparando o resultado obtido com os programas *2Colunas_Contiguidade* e *2Colunas_Não Contiguidade*, constacta-se que o número de casos de independência diminui em *17* unidades e os casos de dependência aumentam na mesma proporção, em relação ao primeiro programa. Em relação ao segundo programa, verifica-se uma diminuição em *16* unidades nos casos de independência e um aumento dos casos de dependência, também em *16* unidades. Verifica-se, ainda, que o primeiro caso de independência, obtido nos resultados do presente programa, é o único que se mantém em relação aos programas mencionados, o segundo caso não se verifica nos mesmos.

Em contrapartida, se em vez de se executar o programa para $ncs = 2$, se executar para $ncs = 3$, ou seja, somando três colunas em vez de duas, os casos de independência obtidos são os seguintes:

000000000111000, 000010100001000, 010000000101000, 010000010100000, 010010010000000 e 011000000001000

Os restantes *449* casos possíveis, não mencionados, dizem respeito a casos de dependência.

Da mesma maneira que no caso anterior, quando se diz que 000000000111000 é um caso de independência, pretende-se dizer que as colunas 11, 12 *e* 13 são independentes. Sob um ponto de vista genético, este resultado significa que há ausência de associação entre os marcadores que se encontram nas colunas 11, 12 *e* 13 do ficheiro referido.

Analogamente ao caso anterior, compara-se o resultado obtido com os programas *3Colunas_Contiguidade* e *3Colunas_ Não Contiguidade.* Depois de uma análise dos mesmos constacta-se que o número de casos de independência diminui em *5* unidades e os casos de dependência aumentam na mesma proporção, em relação ao primeiro programa. Em relação ao segundo programa mencionado, verifica-se uma diminuição em *4* unidades nos casos de independência e um aumento dos casos de dependência, também em *4* unidades. Verifica-se, ainda, que os casos de independência observados nos programas mencionados não se mantêm, isto é, os casos de independência do presente programa não se verificam nos programas citados.

3.10 Soma das Diferenças nos Extremos.py

Neste programa, à semelhança do anterior, pretende-se testar a dependência e independência de colunas, cuja soma de valores nas mesmas é imposta. Todavia, a realização deste programa tem como objectivo a comparação das somas observadas e esperadas nos extremos, isto é, das somas menor e maior observadas e esperadas.

Note-se que os passos seguidos na execução do presente programa são idênticos aos do programa anterior, pelo que apenas serão descritos pormenorizadamente os passos diferentes.

Começa-se por chamar e executar o ficheiro *Dados.py*, *chamar*, visto este conter os dados necessários à execução do programa.

```
# Chamar o ficheiro Dados.py, que contém os dados necessários à execução do programa.

chamar = open("Dados.py")
exec chamar
```

Posteriormente, insere-se o número de colunas que se pretendem somar, *ncs*, seguindo-se a elaboração das funções *suc(w)*, *inv(w)* e *est(L)*, que nos dão, respectivamente, a lista sucessora de uma lista *w*, a lista inversa da mesma e a transformação de uma lista *L* numa palavra. Os passos seguidos na realização das funções são os descritos no programa anterior. As listas *pali* e *palf* representam, respectivamente, a lista inicial e a final, e uma vez explicadas no programa anterior, não serão descritas novamente.

```
# ncs indica o número de colunas a somar

ncs = int(raw_input("insira o número de colunas a somar:\n"))


# suc(w) é a função que nos dá o sucessor de w

def suc( w ):
    for j in range(len(w)):
        if ((w[len(w)-j-1] == 1) and (w[len(w)-j-2] == 0)):
            w[len(w)-j-2] = 1
            w[len(w)-j-1] = 0
            r = 0
            for k in range(len(w)-j,len(w)):
                if (w[k] == 1):
                    r = r+1
            for kk in range(len(w)-r,len(w)):
                w[kk] = 1
```

```
            for kkk in range(len(w)-j,len(w)-r):
                w[kkk] = 0
            break
    return w


# pali é uma lista que representa a palavra inicial

pali = []
for j in range(nc_nao,nc-ncs):
    pali = pali + [0]
for j in range(nc-ncs,nc):
    pali = pali + [1]
#print "pali =", pali


# inv(w) é a função que nos dá o inverso de w
# palf é a lista que representa a palavra inversa de pali

def inv( w ):
    v = []
    for i in range(len(w)):
        v = v + [w[len(w)-i-1]]
    return v
palf = inv( pali )
#print "palf =", palf


# est(L) é a função que transforma uma lista numa palavra

def est( L ):
    u = ''
    for i in range(len(L)):
        u = u + str(L[i])
```

return u

De seguida, determinam-se o número de linhas observadas no ficheiro que têm um determinado valor para a soma, *soma*, das *ncs* colunas consideradas. Da mesma forma que no programa anterior, o valor encontrado será armazenado na lista *dado[est(w),soma]* e uma vez que os passos seguidos são idênticos, estes não serão explicados pormenorizadamente, sendo descritos apenas os que diferem.

O dicionário *dadototal* armazena as listas da forma *dadototal[est(w),soma]*, que indicam o número de linhas observadas no ficheiro original com um determinado valor para a soma, *soma*, das *ncs* colunas representadas por *1* na lista *w*, tendo em conta o número total de linhas do ficheiro.

São consideradas duas variáveis auxiliares, inicialmente nulas, aux_smin e aux_smax, e para cada valor entre *somainicial*, inclusive, e *somainicial+ncs*, esclusive, *smin* adiciona-se ao valor aux_smin o valor *dado[est(w),smin]*, correspondente, denotando o valor obtido por aux_smin. Depois de percorrer todos os valores, o resultado obtido será armazenado na lista $dado_min[est(w)]$, e indicará a soma dos menores valores observados nas colunas consideradas. De forma idêntica, para cada valor entre *somafinal-ncs+1* e *somafinal* inclusive, *smax*, adiciona-se ao valor aux_smax o valor *dado[est(w),smax]* correspondente, denotando o valor obtido por aux_smax, sendo o valor final, isto é, depois de percorridos todos os valores referidos, armazenado na lista $dado_max[est(w)]$, que indicará a soma dos maiores valores observados nas colunas consideradas.

dado[est(w),soma] indica o número de linhas observadas cuja soma das colunas indicadas por 1 em w é soma.
dadototal[est(w),soma] indica o número de linhas observadas cuja soma das colunas indicadas por 1 em w é soma, tendo em conta o número total de linhas do ficheiro.
dado_min[est(w)] indica a soma dos valores dado, no caso das somas minimas.
dado_max[est(w)] indica a soma dos valores dado, no caso das somas máximas.

dado = {}
dadototal = {}

```
dado_min = {}
dado_max = {}
somalinha = {}
w = [] + pali
#print "w =", w
while (0 == 0):
    somainicial = 0
    somafinal = 0
    for j in range(len(w)):
        if w[j] == 1:
            somainicial = somainicial + coluna[j+2][0]
            somafinal = somafinal + coluna[j+2][valores[j+2]-1]
    #print "somainicial", somainicial
    #print "somafinal", somafinal
    for soma in range (somainicial,somafinal+1):
        h = 0
        for i in range(nl_nao,nl):
            sl = 0
            for j in range(len(w)):
                if w[j] == 1:
                    sl = sl + int(t[i][j+2])
            somalinha[i] = sl
            #print "somalinha[", i, "] =", somalinha[i]
            if (somalinha[i] == soma):
                h = h + 1
        dado[est (w ),soma] = h
        #print "dado[", est ( w ), ",", soma, "] =", dado[est( w ),soma]
        dadototal[est( w ),soma] = dado[est( w ),soma] / (nl-nl_nao)
        #print "dadototal["est( w ), ",", soma, "] =", dadototal[est( w ),soma]
```

```
    aux_smin = 0
    aux_smax = 0
    for smin in range(somainicial,somainicial+ncs):
        aux_smin = aux_smin + dado[est (w ),smin]
    for smax in range(somafinal-ncs+1,somafinal+1):
        aux_smax = aux_smax + dado[est (w ),smax]
    dado_min[est(w)] = aux_smin
    dado_max[est(w)] = aux_smax
    #print "dado_min[",est(w),"] =", dado_min[est(w)]
    #print "dado_max[",est(w),"] =", dado_max[est(w)]
    if w == palf:
        break
    else:
        ww = suc(w)
        w = ww
```

Nos passos que se seguem, é determinada a probabilidade de ter um valor l na coluna j, *probcoluna[l,j]*, e as funções *men(w,k)* e *seg(u)*, que nos permitem distribuir, da menor forma, os k valores pelas colunas representadas por *1* em w, e determinar a lista sucessora da lista u, com soma k, nas mesmas colunas, respectivamente. Salienta-se, ainda, que os passos seguidos se encontram descritos no programa anterior, pelo que não serão referidos novamente.

```
# probcoluna[l,j] representa a probabilida de ter o valor l na coluna j.
probcoluna = {}
for j in range(nc_nao,nc):
    for l in range(lmin,lmax+1):
        probcoluna[l,j] = obscoluna[l,j]/(nl-nl_nao)
        #print "probcoluna[", l, ",", j, "] =", probcoluna[l,j]
```

```
# men(w,k) define a menor distribuição de k elementos pelas colunas indicadas por 1 em
w.

def men(w,k):
    u = []
    kk = k
    for j in range(len(w)):
        if (w[j] == 0):
            u = u + [0]
        else:
            u = u + [coluna[nc+j-len(w)][0]]
            kk = kk - coluna[nc+j-len(w)][0]
    if (kk < 0):
        return []
    else:
        for i in range(len(w)):
            if (w[len(w)-i-1] == 1):
                vv = coluna[nc-1-i][valores[nc-1-i]-1]
                if (kk <= vv - u[len(w)-i-1]):
                    u[len(w)-i-1] = u[len(w)-i-1] + kk
                    break
                else:
                    kk = kk - vv + u[len(w)-i-1]
                    u[len(w)-i-1] = vv
        return u


# seg(u) é a função que nos dá o sucessor de u, com soma k, nas colunas indicadas por 1
em w.

def seg(u):
    w = []
```

```
for i in range(len(u)):
    if (u[i] == 0):
        w = w + [0]
    else:
        w = w + [1]
xx = 0
for j in range(len(u)):
    if (w[len(u)-j-1] == 1):
        if (u[len(u)-j-1] > coluna[len(u)-j+1][0]):
            xxx = 0
            for j2 in range(j+1, len(u)):
                if (w[len(u)-j2-1] == 1):
                    if (u[len(u)-j2-1] < coluna[len(u)-j2+1][valores[len(u)-j2+1]-1]):
                        aux1 = 0
                        aux2 = 0
                        for jj in range(len(u)-j2,len(u)):
                            aux1 = aux1 + u[jj]
                            if u[jj] != 0:
                                aux2 = aux2 + coluna[jj+2][0]
                        u[len(u)-j-1] = u[len(u)-j-1] + 1
                        ww = w[len(u)-j : len(u)]
                        kl = aux1 - 1
                        uu = men(ww,kl)
                        for jjj in range(len(uu)):
                            u[jjj+len(u)-j] = uu[jjj]
                            xx = 1
                            xxx = 1
                        break
            if xxx == 1:
```

```
            break
    if xx == 1:
        return u
    else:
        return []
```

Em seguida, é determinado o número de linhas esperadas no ficheiro com um determinado valor para a soma, *soma*, das *ncs* colunas consideradas. O valor determinado será armazenado na lista *esp[est(w),soma]*.

De forma análoga à determinação do número de linhas observadas, nas mesmas condições, começa-se por considerar a lista *pw*, como sendo a lista inicial, *pali*, e inicia-se um ciclo que apenas será interrompido quando *pw* passar a ser a lista final, *palf*. É de notar que os passos seguidos são idênticos ao do programa anterior, pelo que apenas se descreverá os que diferem.

Da mesma maneira que na determinação do número de linhas observadas, consideram-se duas variáveis auxiliares, inicialmente nulas, *aux_smin* e *aux_smax*, e para cada valor entre *somainicial*, inclusive, e *somainicial+ncs*, exclusive, *smin* adiciona-se ao valor *aux_smin* o valor *esp[est(pw),smin]*, correspondente, denotando o valor obtido por *aux_smin*. Depois de percorrer todos os valores, o resultado obtido será armazenado na lista *esp_min[est(pw)]*, e indicará a soma dos menores valores esperados nas colunas consideradas. De forma idêntica, para cada valor entre *somafinal-ncs+1* e *somafinal* inclusive, *smax*, adiciona-se ao valor *aux_smax* o valor *esp[est(pw),smax]* correspondente, denotando o valor obtido por *aux_smax*, sendo o valor final armazenado na lista *est_max[est(pw)]*, que indicará a soma dos maiores valores esperados nas colunas consideradas.

```
# esp[esp(w),soma] indica o número esperado de linhas cuja soma das colunas indicadas por 1 em w é soma, admitindo indep
# entre as colunas.
# esp_min[est(w)] indica a soma dos valores esp, no caso das somas minimas.
# esp_max[est(w)] indica a soma dos valores esp, no caso das somas máximas.
```

```
esp = {}
esp_min = {}
esp_max = {}
pw = [] + pali
while (0 == 0):
    somainicial = 0
    somafinal = 0
    for j in range(len(pw)):
        if (pw[j] == 1):
            somainicial = somainicial + coluna[j+2][0]
            somafinal = somafinal + coluna[j+2][valores[j+2]-1]
    #print "somainicial", somainicial
    #print "somafinal", somafinal
    pt = 0.0
    for soma in range(somainicial,somafinal+1):
        #print "soma =", soma
        u = men(pw,soma)
        #print "u =", u
        p = 0.0
        while (u != []):
            h = 1.0
            for jj in range(len(u)):
                if (u[jj] != 0):
                    h = h*probcoluna[u[jj],jj+2]
            p = p + h
            pt = pt + h
            useg = seg(u)
            u = useg
            #print "u =", u
```

```
        esp[est(pw),soma] = p*(nl-nl_nao)
        #print "esp[", est(pw), ",", soma, "] =", esp[est(pw),soma]
    aux_smin = 0
    aux_smax = 0
    for smin in range(somainicial,somainicial+ncs):
        aux_smin = aux_smin + esp[est (pw ),smin]
    for smax in range(somafinal-ncs+1,somafinal+1):
        aux_smax = aux_smax + esp[est (pw ),smax]
    esp_min[est(pw)] = aux_smin
    esp_max[est(pw)] = aux_smax
    #print est(pw), "min:", "%3.3f"% esp_min[est(pw)], dado_min[est(pw)], "max:", "%3.3f"%
esp_max[est(pw)], dado_max[est(pw)]
    #print est(pw), "min:", esp_min[est(pw)], dado_min[est(pw)], "max:", esp_max[est(pw)],
dado_max[est(pw)]
    #print "esp_min[",est(pw),"] =", esp_min[est(pw)]
    #print "esp_max[",est(pw),"] =", esp_max[est(pw)]
    #print "dado_min[",est(pw),"] =", dado_min[est(pw)]
    #print "dado_max[",est(pw),"] =", dado_max[est(pw)]
    #print ' '
    if pw == palf:
        break
    else:
        z = suc(pw)
        pw = z
        #print "pw =", pw
    #print "pt =", pt
```

Posteriormente, é determinado o valor do *Qui-Quadrado* para cada lista w, seguindo-se a sua comparação com o valor do *Qui-Quadrado* tabelado. Este último passo referido

consiste na realização do teste do *Qui-Quadrado*, que resultará em dependência no caso do primeiro valor ser maior ou igual que o segundo, e independência caso contrário.

```
# Qui[est(w)] representa a estatística do teste Qui-Quadrado, para as colunas assinaladas
com 1 em w.

Qui = {}
w = [] + pali
#print"w =", w
while (0 == 0):
    somainicial = 0
    somafinal = 0
    for j in range(len(w)):
        if (w[j] == 1):
            somainicial = somainicial + coluna[j+2][0]
            somafinal = somafinal + coluna[j+2][valores[j+2]-1]
    #print "somainicial", somainicial
    #print "somafinal", somafinal
    Q = 0.0
    for soma in range (somainicial,somafinal+1):
        Q = Q + (((dado[est(w),soma]-esp[est(w),soma])**2)/esp[est(w),soma])
    Qui[est(w)] = Q
    #print "Qui[", est(w), "]=", Qui[est(w)]
    if w == palf:
        break
    else:
        ww = suc(w)
        w = ww
```

```
# Teste do Qui-Quadrado - verifica se 2 colunas são ou não dependentes.
# gl[est(w)] dá-nos os graus de liberdade entre as colunas assinaladas com 1 em w.
# Qui[est(w)] representa a estatistica do teste Qui-Quadrado, para as colunas assinaladas
com 1 em w.

gl = {}
w = [] + pali
#print"w =", w
while (0 == 0):
    gl[est(w)] = ncs-1
    #print "QQ[", gl[est(w)], "][", 2,"] =", float(QQ[gl[est(w)]][2])
    if (Qui[est(w)] >= float(QQ[gl[est(w)]][2])):
        print "As colunas assinaladas com 1 em", est(w), "são dependentes"
    else:
        print "As colunas assinaladas com 1 em", est(w), "são independentes"
    if w == palf:
        break
    else:
        ww = suc(w)
        w = ww
```

Igualmente ao programa descrito anteriormente, ao executar o presente programa, para $ncs = 2$, no que diz respeito a casos de independência, os resultados obtidos são os seguintes:

000000000011000 e 010000100000000

Os restantes *103* casos possíveis, não mencionados, dizem respeito a casos de dependência.

Da mesma forma que no programa anterior, quando se diz que 000000000011000 é um caso de independência, pretende-se dizer que as colunas 12 *e* 13 são independentes, o que em termos de genética, significa que há ausência de associação entre os marcadores representados pelas colunas 12 *e* 13 do ficheiro *YPortugal_Total2.csv*.

Em contrapartida, se em vez de se executar o programa para $ncs = 2$, se executar para $ncs = 3$, como no programa anterior, ou seja, somando três colunas em vez de duas, os casos de independência obtidos são os seguintes:

000000000111000, 000010100001000, 010000000101000, 010000010100000, 010010010000000, e 011000000001000

Os restantes *449* casos possíveis, não mencionados, dizem respeito a casos de dependência.

Da mesma maneira que no caso anterior, quando se diz que 000000000111000 é um caso de independência, pretende-se dizer que as colunas 11, 12 *e* 13 são independentes. Sob um ponto de vista genético, este resultado significa que há ausência de associação entre os marcadores que se encontram nas colunas 11, 12 *e* 13 do ficheiro referido.

Como se pode constactar, os resultados obtidos no presente programa, para $ncs = 2$ ou $ncs = 3$, são os mesmos do programa anterior.

3.11 Independência de 2 colunas e Sucessivamente.py

Este programa foi elaborado com o intuito de determinar as colunas independentes duas a duas e, posteriormente, apenas para estas colunas, verifica-se a independência três a três, e assim sucessivamente.

É de notar a semelhença deste programa com os programas *2colunas_Contiguidade.py* e *3colunas_Contiguidade.py*. Salienta-se que o presente programa é mais abrangente, no

entanto, apenas os passos que diferem serão pormenorizadamente descritos, assim como exemplificados.

Analogamente aos programas anteriores, começa-se por chamar e executar o ficheiro *Dados.py*. A probabilidade de ter um valor *l*, na coluna *j*, denota-se também por *probcoluna[l,j]*, e determina-se exactamente da mesma forma que no programa *Soma das Diferenças nos Extremos.py*. Note-se que a soma de todas as probabilidades, para cada coluna, é *1*, *p*, como se tem *15* colunas no total, a soma de todas as probabilidades é *15*, *pt*.

```
# Chamar o ficheiro Dados.py, que contém os dados necessários à execução do programa.
chamar = open("Dados.py")
exec chamar


# probcoluna[l,j] representa a probabilidade de ter o valor l na coluna j.
probcoluna = {}
p = 0.0
for j in range(nc_nao,nc):
    pt = 0.0
    for l in range(lmin,lmax+1):
        probcoluna[l,j] = obscoluna[l,j]/(nl-nl_nao)
        #print "probcoluna[", l, ",", j, "] =", probcoluna[l,j]
        p = p + probcoluna[l,j]
        pt = pt + probcoluna[l,j]
    #print "pt =", pt
#print "p =", p
```

No próximo passo constrói-se uma lista com as colunas que se pretendem considerar na execução do programa, *vec*. Começa-se por considerar apenas duas colunas, $n = 2$, e a construção do vector consiste na junção das colunas, ordenadamente, até obter o

comprimento desejado. Note-se que a primeira lista a ser considerada é:

$$vec = [2, 3]$$

que consiste na lista que engloba as colunas *2* e *3*.

```
# vec é a lista que indica as colunas consideradas.
# Começamos com o caso de 2 colunas: n=2.

n = 2
vec = []
k = 0
for j in range(nc_nao,nc):
    if (k < n):
        vec = vec + [j]
        k = k + 1
#print "vec =", vec
```

Em seguida é elaborada uma função, *suc(set)*, que nos dará o sucessor de uma lista *set*. Convém salientar que *set* é uma lista formada pelos valores que se encontram, respectivamente, nas colunas armazenadas na lista *vec*. Um exemplo de uma lista *set*, tendo em conta a lista *vec*, anteriormente citada, será:

$$set = [13, 10]$$

onde *13* é um valor da coluna *2* e *10* é um valor da coluna *3*.

Salienta-se, também, que esta lista *set* é a primeira a ser considerada, tendo em conta este *vec*.

Na função que é apresentada em seguida, começa-se a percorrer a lista *set*, do fim para o incício da mesma, verificando se o valor em causa é menor que o maior valor

que a coluna correspondente pode tomar. Em caso afirmativo, acrescenta-se um valor a essa posição. As posições posteriores passam a tomar o menor valor possível na coluna correspondente, e as posições anteriores permanecem inalteradas. No caso da listas *set* apresentada anteriormente a sua sucessora seria:

$$suc(set) = [13, 11]$$

No caso da lista *set* considerada ser:

$$set = [14, 15]$$

onde *15* é o máximo valor que a coluna *3* pode tomar, a sua lista sucessora seria:

$$suc(set) = [15, 10]$$

Note-se que o valor que aumentou uma unidade foi o correspondente à coluna *2*, visto o valor referente à coluna *3* ser o maior valor possível na mesma, *15*. O valor desta última coluna passou a ser o menor valor possível na mesma, *10*.

```
# suc(set) é a função que nos dá o sucessor do conjunto set.

def suc( set ):
    xx = 0
    for j in range(len(set)):
        if (set[len(set)-j-1] < coluna[vec[len(set)-j-1]][len(coluna[vec[len(set)-j-1]])-1]):
            set[len(set)-j-1] = set[len(set)-j-1] + 1
            for k in range(len(set)-j,len(set)):
                set[k] = coluna[vec[k]][0]
            break
        else:
            if (j == len(set)-1):
```

```
            xx = 1
    if (xx == 1):
        return []
    else:
        return set
```

Posteriormente é apresentada outra função que nos dará o sucessor de uma lista *vec*, onde *vec*, como já foi referido, é a lista formada pelas colunas que se estão a considerar.

Da mesma forma que na função anterior, percorre-se a lista do fim para o início da mesma, e verifica-se se a coluna, na posição que se está a analisar, coincide com a última coluna do ficheiro. Se ainda for menor, então aumenta-se uma unidade, passando a representar a coluna seguinte. As posições posteriores da lista passam a ser as colunas posteriores a esta, de forma ordenada, e as anteriores permanecem inalterads. No caso da lista *vec* apresentada anteriormente, a sua sucessora seria:

$$sucessor(vec) = [2, 4]$$

```
# sucessor(vec) é a função que nos dá o sucessor de vec.
def sucessor( vec ):
    xx = 0
    for j in range(len(vec)):
        if (vec[len(vec)-j-1] < nc-1-j):
            vec[len(vec)-j-1] = vec[len(vec)-j-1] + 1
            for k in range(len(vec)-j,len(vec)):
                vec[k] = vec[k-1]+1
            break
        else:
```

```
            if (j == len(vec)-1):
                xx = 1
    if (xx == 1):
        return []
    else:
        return vec
```

A próxima função permite transformar uma lista numa palavra, na qual os diferentes elementos da lista se encontram separados por vírgulas, cujo objectivo é a distinção dos mesmos. Assim, com esta finalidade, pela mesma ordem com que os elementos aparecem na lista, antes de se chegar ao último, procede-se à sua junção com a lista u, inicialmente vazia, seguido-o de uma virgula, sendo cada uma das junções denotada novamente por u. Note-se que a lista u está a ser constantemente actualizada. Por exemplo, para a lista *vec* referida acima, *est(vec)* seria:

$$est(vec) = 2, 3$$

```
# est(L) é a função que transforma uma lista numa palavra, cujos elementos da palavra
estão separados por vírgulas.

def est( L ):
    u = ''
    for i in range(len(L)):
        u = u + str(L[i])
        if (i != len(L)-1):
            u = u + ','
    return u
#print "est(",vec,") =", est(vec)
```

Nos próximos passos pretende-se agrupar as colunas que são independentes, n a n, *Ind[n]*. Anteriormente foi definido n=*2*, pelo que se começa por ver as colunas independentes *2* a *2*, só depois, tendo em conta apenas as colunas agrupadas nesta lista, verifica-se a independência *3* a *3* e assim sucessivamente. Note-se que se pressupõe, à priori, *Ind[n]=[]*, para cada n.

De forma idêntica aos programas já mencionados anteriormente, inicia-se um ciclo, o qual só será interrompido no caso da lista *sucessor(vec)* ser a lista nula. Recorde-se que a lista *vec* é uma lista formada pelas colunas com que se pretendem trabalhar.

Tal como já foi referido, este programa é bastante semelhante aos programas *2colunas_Contiguidade.py* e *3colunas_Contiguidade.py* pelo que os passos apresentados não serão pormenorizadamente explicados, à excepção dos que diferem.

Tendo em conta a lista *vec* criada no inicio do programa, começa-se por criar uma lista *set*, cujas componentes são os elementos mínimos das colunas presentes em *vec*.

Posteriormente, inicia-se outro ciclo, no qual se determina o número de linhas esperadas com os valores da lista *set*, nas respectivas colunas armazenadas na lista *vec*, *Q[i]*, onde a variável i indica a i-ésima lista *set* referente à lista *vec* em causa. Considera-se ainda uma lista *D[i]* que coincide com a lista *set*, e com a qual se trabalhará neste ciclo. Note-se que cada lista *Q[i]* consiste no produto das probabilidades de ter os valores da lista *D[i]* nas respectivas colunas guardadas na lista *vec*, pelo número total de linhas do ficheiro *YPortugal_Total2.csv*.

Em seguida, determina-se o número de linhas observadas no ficheiro, nas mesmas condições. Contam-se o número de linhas que têm os elementos da lista *set*, nas respectivas colunas representadas pelas posições da lista *vec*, *dado[i]*. A lista *dadototal[i]* representa o número de linhas observadas, tendo em conta o número total de linhas.

Depois destes passos determina-se a lista sucessora da lista *set*, *novo_set*. Se esta for diferente da lista vazia, então substitui a lista *set* anterior, e a variável i aumenta uma unidade, e o ciclo repete-se, caso contrário é interrompido, e o ciclo inicial prossegue.

set é a lista que indica os valores presentes nas colunas consideradas, pela mesma ordem que estas aparecem em vec.

```
# D[i] é a lista que representa os valores presentes nas colunas consideradas, ie, set.
# Q[i] é a lista que representa o número de linhas esperado com os valores de set nas colunas de vec.
# dado[D[i]] indica o número de linhas observadas com os valores de set nas colunas de vec.
# dadototal[D[i]] indica o número de linhas observadas com os valores de set nas colunas de vec, tendo em conta o número total de linhas do ficheiro.
# cont[x,y] é a lista que indica se D[x] e D[y] são contíguos (= 1) ou não (= 0).
# Listas D e Q, após a junção dos pequeninos contíguos entre si e junção dos pequeninos contíguos, ao grande mais próximo.
# Qui[j,jj] representa a estatística do teste Qui-Quadrado, para as colunas j e jj.
# gl[j,jj] dá-nos os graus de liberdade entre as colunas j e jj.
# Ind[n] é a lista cujos elementos são as colunas independentes n a n.

Ind = {}
Ind[n] = []
while (0 == 0):
    dado = {}
    dadototal = {}
    i = 0
    D = {}
    Q = {}
    Qui = {}
    gl = {}
    g2c = {}
    cont = {}
    set = []
    for ii in range(len(vec)):
        set = set + [coluna[vec[ii]][0]]
    #print "set =", set
```

```
while ( 0 == 0):
    D[i] = [] + set
    #print "D[", i, "] =", D[i]
    prob = 1.0
    for l in range(len(vec)):
        prob = prob*probcoluna[D[i][l],vec[l]]
    Q[i] = prob*(nl-nl_nao)
    #print "Q[", i, "] =", Q[i]
    h = 0.0
    for lin in range(nl_nao,nl):
        bp = 0
        for j in range(len(vec)):
            if (int(t[lin][vec[j]]) == D[i][j]):
                continue
            else:
                bp = 1
                break
        if (bp == 0):
            h = h + 1.0
    #print "h =", h
    dado[i] = h
    #print "dado[", i, "] =", dado[i]
    dadototal[i] = dado[i] / (nl-nl_nao)
    #print "dadototal[", i , "] =", dadototal[i]
    novo_set = suc(set)
    if (novo_set == []):
        break
    else:
        set = novo_set
```

```
#print "set =", set
i = i + 1
```

Nos passos posteriores, e tendo em conta que não faz sentido considerar valores esperados menores que 5, agrupam-se os valores do domínio, contíguos, com valor esperado menor que 5, por ordem crescente, até serem maiores que *5*. No caso de não dar para agrupar mais os valores, e se continuar a ter valores esperados menores que *5*, então estes serão agrupados por ordem decrescente ao valor do dominio mais próximo com valor esperado maior que *5*. Os passos referidos são descritos em seguida.

Começa-se por agrupar os elementos do domínio de *D*, *Q*, *dado* e *dadototal*, cujo valor de *Q[i]* seja menor que *5*. Os passos seguidos são idênticos aos dos programas referidos, pelo que não se justificará a sua descrição. Note-se, apenas, que se trabalha com listas *set*, *D[i]* e *vec* com o mesmo comprimento, n, que de inicio toma o valor *2*, pelo que se percorrem os elementos dessas listas em passos de n.

Dê-se especial atenção à determinação da condição de contiguidade (caso em que a mesma é igual a *1*). Esta é determinada da seguinte forma:

$$\sum_{e=0}^{n-1}(D[z][p+e] - D[tt][pp+e])^2$$

onde z e tt são elementos do domínio do dicionário D e p e pp são as variáveis que percorrem os elementos das listas *D[z]* e *D[tt]*, respectivamente, em passos de n.

De seguida, tendo em conta as mesmas condições do programas mencionados, descreva-se, z e tt contíguos ($cont[z, tt] = 1$) e $Q[tt] < k - h$, actualizam-se os dicionários *D*, *Q*, *dado* e *dadototal*, exactamente da mesma forma.

```
w = 0
while (w == 0):
    w = 1
    for z in D.keys():
```

```
C = 0
ww = 0
for p in range(0,len(D[z]),n):
    if (C == 1):
        break
    else:
        for k in range(2,8):
            for h in range(int(k/2),0,-1):
                #print "Q[", z, "] =", Q[z]
                #print "h =", h
                if (Q[z] < h):
                    for tt in D.keys():
                        if (tt != z):
                            for pp in range(0,len(D[tt]),n):
                                soma_aux = 0
                                for e in range(n):
                                    soma_aux = soma_aux + (D[z][p+e]-D[tt][pp+e])**2
                                #print "soma_aux =", soma_aux
                                if (soma_aux == 1):
                                    C = 1 # ié, D[z] contíguo com D[t]
                                    break
                                else:
                                    C = 0
                            cont[z,tt] = C
                            #print "cont[", z, ",", t, "] =", cont[z,t]
                            if ((cont[z,tt] == 1) and (Q[tt] < k-h)):
                                w = 0
                                D[z] = D[z] + D[tt]
                                Q[z] = Q[z] + Q[tt]
```

```
                    dado[z] = dado[z] + dado[tt]
                    dadototal[z] = dadototal[z] + dadototal[tt]
                    del D[tt]
                    del Q[tt]
                    del dado[tt]
                    del dadototal[tt]
                    ww = 1
                    break
        if (ww == 1):
            break
#print "D =", D
#print "Q =", Q
#print "dado =", dado
#print "dadototal =", dadototal
```

Em seguida, são agrupados os valores do domínio com valor esperado menor que *5*, ao valor mais próximo com valor esperado maior que *5*.

O processo adoptado para agrupar os valores segue um raciocínio idêntico ao usado nos programas mencionados, assim como do ciclo anterior, pelo que não faz sentido voltar a descrever.

```
for k in range(5,0,-1):
    H = 0
    while (H == 0):
        H = 1
        for x in D.keys():
            C = 0
            ww = 0
```

```
for p in range(0,len(D[x]),n):
    if (C == 1):
        break
    else:
        if ((Q[x] < k) and (Q[x] >= k-1)):
            z = 99
            for y in D.keys():
                if (y != x):
                    for pp in range(0,len(D[y]),n):
                        soma_aux = 0
                        for e in range(n):
                            soma_aux = soma_aux + (D[x][p+e]-D[y][pp+e])**2
                        if (soma_aux == 1):
                            C = 1 # ié, D[x] contíguo com D[y]
                            break
                        else:
                            C = 0
                    cont[x,y] = C
                    if (cont[x,y] == 1):
                        if (z == 99):
                            z = y
                        else:
                            if (Q[y] < Q[z]):
                                z = y
                    D[x] = D[x] + D[y]
                    Q[x] = Q[x] + Q[y]
                    dado[x] = dado[x] + dado[y]
                    dadototal[x] = dadototal[x] + dadototal[y]
                    del D[y]
```

```
                    del Q[y]
                    del dado[y]
                    del dadototal[y]
                    H = 0
                    ww = 1
                    break
            if (ww == 1):
                break
    #print "D =", D
    #print "Q =", Q
    #print "dado =", dado
    #print "dadototal =", dadototal
```

Posteriormente, para a lista *vec* em causa calcula-se o valor do *Qui-Quadrado*, *Qui[est(vec)]*, que se determina da seguinte forma:

$$\sum_{x \in D.keys} \frac{(dado[x] - Q[x])^2}{Q[x]}$$

Seguidamente determinam-se os graus de liberdade, que consiste no número de elementos presentes no dicionário D menos uma unidade.

Para terminar, realiza-se o teste do *Qui-Quadrado*, comparando o valor do *Qui-Quadrado* calculado anteriormente, *Qui[vec]*, com o valor tabelado, *QQ[gl[est(vec)]][2]* (note-se que a linha é dada pelos graus de liberdade referentes à lista *vec* em causa e a coluna é a *2*, *probabilidade* = 0.05). No caso do primeiro ser maior ou igual que o segundo, as colunas armazenadas na lista *vec* são dependentes, $g2c[est(vec)] = 1$, caso contrário são independentes, $g2c[est(vec)] = 0$. Se para a lista *vec* em causa se verificar este último caso, isto é, as colunas serem independentes, então a lista *vec* será armazenada na lista *Ind[n]*, e passa-se à determinação da sua lista sucessora, *novo_vec*. Caso esta não

seja nula então denota-se por *vec*, substituindo o vec anterior, e o ciclo repete-se, caso contrário, o ciclo é interrompido.

Depois de terminado este ciclo, a lista *Ind[n]* conterá todas as listas compostas por n colunas independentes entre si. Note-se que neste primeiro ciclo apenas se determinou *Ind[2]*, isto é, conjuntos de *2* colunas independentes.

```
var = 0.0
for x in D.keys():
    var = var + (((dado[x]-Q[x])**2)/Q[x])
Qui[est(vec)] = var
#print "Qui[",est(vec), "] =", Qui[est(vec)]
gl[est(vec)] = int(len(D.keys())-1)
#print "gl[",est(vec), "] =", gl[est(vec)]
# Teste do Qui-Quadrado - verifica se 2 colunas são ou não dependentes:
#print "QQ[", gl[est(vec)], "][2] =", float(QQ[gl[est(vec)]][2])
if (Qui[est(vec)] >= float(QQ[gl[est(vec)]][2])):
    g2c[est(vec)] = 1
    #print "As colunas", vec, "são dependentes"
else:
    g2c[est(vec)] = 0
    #print "As colunas", vec, "são independentes"
#print "g2c =", g2c
if (g2c[est(vec)] == 0):
    Ind[n] = Ind[n] + [[] + vec]
#print "Ind[", n, "] =", Ind[n]
novo_vec = sucessor(vec)
if (novo_vec == []):
    break
else:
```

```
        vec = novo_vec
        #print "vec =", vec
print "Ind[", n, "] =", Ind[n]
```

Como já foi referido, depois de terminado este ciclo, o resultado será:

$$Ind[2] = [[2,13],[2,16],[3,4],[3,13],[4,11],[4,12],[4,16],[5,12],[5,13],[6,13],$$
$$[8,11],[8,12],[8,13],[11,13],[11,16],[12,13],[13,14],[13,15],[13,16]]$$

No próximo ciclo determinam-se as listas *Ind[n]*, com $n > 2$, isto é, as listas compostas pelos conjuntos de colunas independentes n a n, mas com $n > 2$. Neste ciclo usar-se-á a lista *Ind[2]* determinada anteriormente, o que justifica o facto desta ter sido determinada primeiramente.

Dá-se inicio a um novo ciclo, o qual só é percorrido quando a lista *Ind[n]* for diferente da lista vazia. Neste caso, considera-se uma nova lista, *Ind[n+1]*, inicialmente igual à lista vazia, e enquanto a lista *Ind[n]* for diferente da nula, percorrem-se todos os seus elementos (note-se que os elementos de *Ind[n]* são também listas, as quais contêm as colunas que são independentes n a n).

Começa-se então por tomar cada uma das listas que compoêm a lista *Ind[n]*, denotando-a por *vector*. Posteriormente, consideram-se, apenas, as colunas posteriores à coluna presente na última posição da lista *vector*, denotando a coluna que está a ser considerada por k. Em seguida, para cada uma das posições de *vector*, tendo em conta a coluna k que se tomou anteriormente, considera-se uma lista formada pelas restantes posições de *vector* acrescentada da coluna k, *novo_vector*, e verifica-se se esta nova lista se encontra na lista *Ind[n]*, isto é, se as n colunas que constam em *novo_vector* são independentes. Caso esta condição seja satisfeita, passa-se à próxima posição de *vector*, e repecte-se o processo. Se, para cada uma das posições de *vector*, tendo em conta a coluna k, cada um dos *novo_vector* pertencer a *Ind[n]*, então à lista *vector* anterior agrupa-se a coluna k, denotando a nova lista por *vec*.

```
# novo_vector é um dicionário formado por listas compostas pelas posições de vector, à
excepção de uma, com as colunas posteriores à última.

novo_vector = {}
while (0 == 0):
    if (Ind[n] == []):
        break
    else:
        Ind[n+1] = []
        while (Ind[n] != []):
            for j in range(len(Ind[n])):
                vector = Ind[n][j]
                #print "vector =", vector
                for k in range(vector[len(vector)-1]+1,nc):
                    #print "k =", k
                    c = 0
                    for i in range(len(vector)):
                        #print "i =", i
                        vector1 = vector[0:i]
                        #print "vector1 =", vector1
                        vector2 = vector[i+1:len(vector)]
                        #print "vector2 =", vector2
                        novo_vector = vector1 + vector2 + [k]
                        #print "novo_vector =", novo_vector
                        if (novo_vector in Ind[n]):
                            #print 'true'
                            if (i == len(vector)-1):
                                c = 2
                                #print "c =", 2
                            else:
```

```
            continue
        else:
            c = 1
            break
    #print "c =", c
    if (c == 2):
        vector = vector + [k]
        vec = vector
        #print "vec =", vec
```

Posteriormente, tendo em conta a lista *vec* formada anteriormente, procede-se de forma análoga à anterior, verificando se as colunas presentes em *vec* são independentes ou não. Como os passos seguidos são idênticos aos anteriores, à excepção de n, que agora será substituído por $n + 1$, não será necessário descrevê-los novamente.

Depois de determinar as colunas independentes $n+1$ a $n+1$, estas são armazenadas na lista *Ind[n+1]*, analogamente ao ciclo anterior. Posteriormente, faz-se $n = n + 1$, e o ciclo repete-se, até que *Ind[n]* seja a lista nula, como já foi referido.

```
dado = {}
dadototal = {}
i = 0
D = {}
Q = {}
Qui = {}
gl = {}
g2c = {}
cont = {}
set = []
```

```
#print "set =", set
for ii in range(len(vec)):
   set = set + [coluna[vec[ii]][0]]
#print "set =", set
while ( 0 == 0):
   D[i] = [] + set
   #print "D[", i, "] =", D[i]
   prob = 1.0
   for l in range(len(vec)):
      prob = prob*probcoluna[D[i][l],vec[l]]
   Q[i] = prob*(nl-nl_nao)
   #print "Q[", i, "] =", Q[i]
   h = 0.0
   for lin in range(nl_nao,nl):
      bp = 0
      for j in range(len(vec)):
         if (int(t[lin][vec[j]]) == D[i][j]):
            continue
         else:
            bp = 1
            break
      if (bp == 0):
         h = h + 1.0
   #print "h =", h
   dado[i] = h
   #print "dado[", i, "] =", dado[i]
   dadototal[i] = dado[i] / (nl-nl_nao)
   #print "dadototal[", i , "] =", dadototal[i]
   novo_set = suc(set)
```

```
    if (novo_set == []):
        break
    else:
        set = novo_set
        #print "set =", set
        i = i + 1
w = 0
while (w == 0):
    w = 1
    for z in D.keys():
        C = 0
        ww = 0
        for p in range(0,len(D[z]),n+1):
            if (C == 1):
                break
            else:
                for k in range(2,8):
                    for h in range(int(k/2),0,-1):
                        #print "Q[", z, "] =", Q[z]
                        #print "h =", h
                        if (Q[z] < h):
                            for tt in D.keys():
                                if (tt != z):
                                    for pp in range(0,len(D[tt]),n+1):
                                        soma_aux = 0
                                        for e in range(n+1):
                                            soma_aux = soma_aux + (D[z][p+e]-D[tt][pp+e])**2
                                        #print "soma_aux =", soma_aux
                                        if (soma_aux == 1):
```

```
                    C = 1 # ié, D[z] contíguo com D[t]
                    break
                else:
                    C = 0
            cont[z,tt] = C
            #print "cont[", z, ",", tt, "] =", cont[z,tt]
            if ((cont[z,tt] == 1) and (Q[tt] < k-h)):
                w = 0
                D[z] = D[z] + D[tt]
                Q[z] = Q[z] + Q[tt]
                dado[z] = dado[z] + dado[tt]
                dadototal[z] = dadototal[z] + dadototal[tt]
                del D[tt]
                del Q[tt]
                del dado[tt]
                del dadototal[tt]
                ww = 1
                break
        if (ww == 1):
            break
#print "D =", D
#print "Q =", Q
#print "dado =", dado
#print "dadototal =", dadototal
for k in range(5,0,-1):
    H = 0
    while (H == 0):
        H = 1
        for x in D.keys():
```

```
C = 0
ww = 0
for p in range(0,len(D[x]),n+1):
   if (C == 1):
      break
   else:
      if ((Q[x] < k) and (Q[x] >= k-1)):
         z = 99
         for y in D.keys():
            if (y != x):
               for pp in range(0,len(D[y]),n+1):
                  soma_aux = 0
                  for e in range(n+1):
                     soma_aux = soma_aux + (D[x][p+e]-D[y][pp+e])**2
                  if (soma_aux == 1):
                     C = 1 # ié, D[x] contíguo com D[y]
                     break
                  else:
                     C = 0
               cont[x,y] = C
               if (cont[x,y] == 1):
                  if (z == 99):
                     z = y
                  else:
                     if (Q[y] < Q[z]):
                        z = y
               D[x] = D[x] + D[y]
               Q[x] = Q[x] + Q[y]
               dado[x] = dado[x] + dado[y]
```

```
                    dadototal[x] = dadototal[x] + dadototal[y]
                    del D[y]
                    del Q[y]
                    del dado[y]
                    del dadototal[y]
                    H = 0
                    ww = 1
                    break
            if (ww == 1):
                break
#print "D =", D
#print "Q =", Q
#print "dado =", dado
#print "dadototal =", dadototal
var = 0.0
for x in D.keys():
    var = var + (((dado[x]-Q[x])**2)/Q[x])
Qui[est(vec)] = var
#print "Qui[",est(vec), "] =", Qui[est(vec)]
gl[est(vec)] = int(len(D.keys())-1)
#print "gl[",est(vec), "] =", gl[est(vec)]
# Teste do Qui-Quadrado - verifica se 2 colunas são ou não dependentes:
#print "QQ[", gl[est(vec)], "][2] =", float(QQ[gl[est(vec)]][2])
if (Qui[est(vec)] >= float(QQ[gl[est(vec)]][2])):
    g2c[est(vec)] = 1
    #print "As colunas", vec, "são dependentes"
else:
    g2c[est(vec)] = 0
    #print "As colunas", vec, "são independentes"
```

```
            #print "g2c =", g2c
            if (g2c[est(vec)] == 0):
               Ind[n+1] = Ind[n+1] + [[] + vec]
            #print "Ind[", n+1, "] =", Ind[n+1]
      #print "Ind[", n+1, "] =", Ind[n+1]
   print "Ind[", n+1, "] =", Ind[n+1]
   n = n+1
   #print "n =", n
   break
```

Ao executar o programa descrito, os resultados obtidos são os seguintes:

$$Ind[2] = [[2,13],[2,16],[3,4],[3,13],[4,11],[4,12],[4,16],[5,12],[5,13],[6,13],[8,11],$$
$$[8,12],[8,13],[11,13],[11,16],[12,13],[13,14],[13,15],[13,16]]$$
$$Ind[3] = [[5,12,13],[8,11,13],[8,12,13],[11,13,16]]$$
$$Ind[4] = []$$

Recorde-se que $Ind[2]$ é a lista onde constam os pares de colunas independentes duas a duas (note-se que os pares de colunas que se encontram nesta lista coincidem com os pares de colunas independentes duas a duas obtidas aquando da execução do programa *2colunas_Contiguidade.py*, uma vez que o programa mencionado é um caso particular do presente programa), $Ind[3]$ é a lista onde constam as triplas de colunas independentes três a três, e assim sucessivamente. No entanto, na formação da lista $Ind[3]$, apenas são armazenadas as triplas de colunas, que quando consideradas duas a duas se encontram armazenadas em $Ind[2]$ (note-se, mais uma vez, que existe semelhança destes resultados com os obtidos no programa *3colunas_Contiguidade*, não constando em $Ind[3]$ apenas as triplas de colunas independentes, que quando consideradas duas a duas são dependentes), e assim sucessivamente. Por exemplo, a tripla de colunas $[5,12,13]$ presente na lista $Ind[3]$

significa que as colunas 5, 12 *e* 13 são independentes, mas as colunas 5 *e* 12, 5 *e* 13 e 12 *e* 13 também o são, quando consideradas duas a duas. Seguindo este raciocínio, como se pode constactar, a lista *Ind*[4] é a lista vazia, o que significa que não existem quatro colunas independentes, que quando consideradas em triplas também o sejam. Em termos genéticos isto significa que é possível encontrar pares de marcadores sem qualquer associação (armazenados em *Ind*[2]) e tendo em conta apenas estes marcadores para um estudo seguinte, é possível obter triplas de marcadores com ausência de associação, também (guardados em *Ind*[3]). No entanto, para um estudo ainda mais avançado, considerando apenas os marcadores seleccionados neste último estudo, já não é possível encontrar marcadores com ausência de associação quando considerados em grupos de quatro (lista *Ind*[4]). Assim, tendo em conta as condições impostas, não seria possível prosseguir este estudo.

No anexo - *Anexo D* - é possível visualizar um esquema do presente programa.

3.12 Um Contra Todos.py

Neste programa pretende-se testar a independência de uma coluna em relação à soma das restantes.

De forma análoga aos programas apresentados anteriormente, começa-se por chamar e executar o ficheiro *Dados.py*.

```
# Chamar o ficheiro Dados.py, que contém os dados necessários à execução do programa.
chamar = open("Dados.py")
exec chamar
```

No próximo ciclo, para cada linha do ficheiro *YPortugal_Total2.csv*, *i*, vai-se considerando cada uma da colunas, *j*, desde *nc_nao*, inclusive, até nc, exclusive, e para cada

uma delas somam-se os valores que constam nas restantes colunas e armazena-se esse valor na lista *mais[i,j]*.

Por exemplo,

$$mais[1,2] = 212$$

significa que a soma de todos os valores da 2^a*linha* (linha 1 em *Python*), à excepção do valor da 3^a*coluna* (coluna 2 em *Pythom*) é 212.

```
# mais[i,j] dá-nos a soma dos valores da linha i, nas colunas diferentes de j.
# mais_mais[j] é a lista que nos dá as somas dos valores de cada linha, nas colunas diferentes
de j.
# mais_mais_max[j] dá-nos o valor máximo das somas de cada linha, nas colunas diferentes
de j.
# mais_mais_min[j] dá-nos o valor minimo das somas de cada linha, nas colunas diferentes
de j.

mais = {}
for i in range(nl_nao,nl):
    for j in range(nc_nao,nc):
        aux = 0
        for jj in range(nc_nao,nc):
            if (jj != j):
                aux = aux + int(t[i][jj])
        mais[i,j] = aux
        #print "mais[", i, ",", j, "] =", mais[i,j]
```

Posteriormente, para cada coluna j, consideram-se todas as linhas do ficheiro à excepção das *nl_nao* primeiras, e armazena-se na mesma lista $mais_mais[j]$ todas as listas $mais[i,j]$ correspondentes.

```
mais_mais = {}
for j in range(nc_nao,nc):
    list = []
    for i in range(nl_nao,nl):
        list = list + [mais[i,j]]
    mais_mais[j] = list
    #print "mais_mais[", j, "] =", mais_mais[j]
```

Em seguida, para cada lista $mais_mais[j]$, calculada anteriormente, determina-se o valor mais alto e o valor mais baixo que se encontra na mesma, $mais_mais_max[j]$ e $mais_mais_min[j]$, respectivamente. Assim, para cada coluna j começa-se por considerar duas variáveis auxiliares, $aux_min = 1000$ e $aux_max = 0$. Posteriormente, para cada posição, i, da lista referida, verifica-se se o valor que se encontra na mesma, $mais_mais[j][i]$, é maior que aux_max ou menor que aux_min. Se for o primeiro caso, então aux_max será substituido por este novo valor, no caso do segundo, será a variável aux_min a ser substituída. Depois de percorridas todas as posições da lista $mais_mais[j]$ em causa, aux_min e aux_max representam, respectivamente, $mais_mais_min[j]$ e $mais_mais_max[j]$.

```
mais_mais_max = {}
mais_mais_min = {}
for j in range(nc_nao,nc):
    aux_min = 1000
    aux_max = 0
    for i in range(len(mais_mais[j])):
        if (mais_mais[j][i] > aux_max):
            aux_max = mais_mais[j][i]
        if (mais_mais[j][i] < aux_min):
            aux_min = mais_mais[j][i]
    mais_mais_max[j] = aux_max
```

```
#print "mais_mais_max[", j, "] =", mais_mais_max[j]
mais_mais_min[j] = aux_min
#print "mais_mais_min[", j, "] =", mais_mais_min[j]
```

No ciclo que se segue determina-se o número de linhas observadas com um determinado valor l na coluna j, e soma das restantes colunas, diferentes de j, igual a *soma*, *dado[l,j,soma]*. Assim, para cada coluna, j, percorrem-se todos os valores diferentes que se podem encontrar na mesma, l, e para cada um deles toma-se um valor para *soma* entre $mais_mais_min[j]$ e $mais_mais_max[j]$, inclusive, isto é, entre a soma mínima e a soma máxima possível nas colunas diferentes da coluna em causa. Seguidamente, para cada posição de $mais_mais[j]$, i, verifica-se se o valor que se encontra nessa posição é igual ao valor *soma* em causa e se o valor que se encontra na respectiva linha $i + nl_nao$ e coluna j coincide com o valor l. No caso destas duas condições se verificarem, aumenta-se um valor à variável h, inicialmente nula.

A lista *dadototal[l,j,soma]* consiste no número de linhas observadas nas condições referidas, tendo em conta o número total de linhas do ficheiro com que se trabalha. Cada uma destas listas determina-se dividindo o valor *dado[l,j,soma]* pelo número de linhas do ficheiro que são consideradas.

Por exemplo,

$$dado[13, 2, 204] = 2.0 \qquad \text{e} \qquad dadototal[13, 2, 204] = 0.00304414003044$$

significam, respectivamente, o número de linhas observadas com o valor 13 na $3^a coluna$ e cuja soma dos valores nas colunas diferentes desta é 204, e o número de linhas observadas, nas mesmas condições, mas tendo em conta o número total de linhas do ficheiro.

```
# dado[l,j,soma] indica o número de linhas observadas com o valor l na coluna j e cuja
soma das colunas diferentes de j é soma.
# dadototal[l,j,soma] indica o número de linhas observadas com o valor l na coluna j e
cuja soma das colunas diferentes de j é soma, tendo em conta o número total de linhas do
ficheiro.
```

```
dado = {}
dadototal = {}
for j in range(nc_nao,nc):
    for l in coluna[j]:
        for soma in range(mais_mais_min[j],mais_mais_max[j]+1):
            h = 0.0
            for i in range(len(mais_mais[j])):
                if ((mais_mais[j][i] == soma) and (int(t[i+nl_nao][j]) == l)): # i+nl_nao,
porque a linha nl_nao não conta.
                    h = h + 1.0
            dado[l,j,soma] = h
            #print "dado[",l, ",", j, ",", soma, "] =", dado[l,j,soma]
            dadototal[l,j,soma] = dado[l,j,soma] / (nl-nl_nao)
            #print "dadototal[", l, ",", j, ",", soma, "] =", dadototal[l,j,soma]
```

Posteriormente, para cada coluna j e cada valor possível na mesma, l, determina-se a probabilidade de ter l em j, *probcoluna[l,j]*, que consiste na razão entre o número de vezes que l se encontra em j, isto é, *obscoluna[l,j]*, e o número total de linhas do ficheiro.

Em seguida, para as coluna diferentes de j, é ainda determinada a probabilidade de ter uma determinada soma, *p[j,soma]*, que consiste na razão entre o número de linhas cuja soma dos valores nas colunas diferente de j é *soma*, e o número total de linhas do ficheiro com que se trabalha. Note-se que o valor de *soma* para cada coluna j encontra-se entre a soma mínima e a soma máxima possível nas colunas diferentes da coluna em causa.

Para terminar, para cada coluna j determina-se a probabilidade de ter um determinado valor l na mesma e uma soma igual a *soma* dos valores presentes nas restantes colunas, *prob[l,j,soma]*, que consiste no produto das duas probabilidades determinadas anteriormente, relembre-se, *probcoluna[l,j]* e *prob[l,j,soma]*.

```
# probcoluna[l,j] representa a probabilidade de ter o valor l na coluna j.
# p[j,soma] dá-nos a probabilidade da soma dos valores das colunas diferentes de j ser igual a soma.
# prob[l,j,soma] dá-nos a probabilidade de ter o valor l na coluna j e a soma dos valores das colunas diferentes de j ser soma.

probcoluna = {}
p = {}
prob = {}
for j in range(nc_nao,nc):
    #p1 = 0.0
    #p2 = 0.0
    #p3 = 0.0
    for l in coluna[j]:
        probcoluna[l,j] = obscoluna[l,j]/(nl-nl_nao)
        #print "probcoluna[", l, ",", j, "] =", probcoluna[l,j]
        #p1 = p1 + probcoluna[l,j]
    for soma in range(mais_mais_min[j],mais_mais_max[j]+1):
        h = 0.0
        for i in range(len(mais_mais[j])):
            if (mais_mais[j][i] == soma):
                h = h + 1.0
        p[j,soma] = h /(nl-nl_nao)
        #print "p[", j, ",", soma, "] =", p[j,soma]
        #p2 = p2 + p[j,soma]
        for l in coluna[j]:
            prob[l,j,soma] = probcoluna[l,j]*p[j,soma]
            #print "prob[",l, ",", j, ",", soma, "] =", prob[l,j,soma]
            #p3 = p3 + prob[l,j,soma]
    #print "p1 =", p1
```

```
#print "p2 =", p2
#print "p3 =", p3
```

No próximo passo determina-se o número esperado de linhas com um valor l na coluna j e uma soma das colunas diferentes de j igual a *soma*, que se armazena na lista *esp[l,j,soma]*. Note-se, primeiramente, que a soma *soma* que se está a considerar, tal como nos casos anteriores, varia entre a soma mínima possível das colunas diferentes de j, $mais_mais_min[j]$ e a soma máxima possível das mesmas, $mais_mais_max[j]$. Cada uma das listas *esp[l,j,soma]* é determinada, multiplicando a probabilidade *prob[l,j,soma]* correspondente, calculada anteriormente, pelo número total de linhas consideradas.

No caso de:

$$esp[13, 2, 204] = 0.89497716895$$

significa que o número esperado de linhas com o valor 13 na 3^a*coluna* e cuja soma dos valores nas colunas diferentes da referida é 204 é 0.89497716895.

```
# esp[l,j,soma] indica o número esperado de linhas com o valor l na coluna j e cuja soma
das colunas diferentes de j é soma, admitindo independência entre as colunas.

esp = {}
for j in range(nc_nao,nc):
    for l in coluna[j]:
        for soma in range(mais_mais_min[j],mais_mais_max[j]+1):
            esp[l,j,soma] = prob[l,j,soma]*(nl-nl_nao)
            #print "esp[", l, ",", j, ",", soma, "] =", esp[l,j,soma]
```

De seguida determina-se o valor do *Qui-Quadrado* para cada coluna j, *Qui[j]*. Este valor determina-se da seguinte forma:

$$\sum_{l \in coluna[j]} \sum_{mais_mais_min[j]}^{mais_mais_max[j]} \frac{(dado[l,j,soma] - esp[l,j,soma])^2}{esp[l,j,soma]}$$

Note-se que, como o valor *esp[l,j,soma]* aparece em denominador no cálculo apresentado anteriormente, pelo que tem de se impor a condição $esp[l,j,soma] \neq 0$.

```
# Qui[j] representa a estatística do teste Qui-Quadrado, da coluna j contra a soma das
colunas diferentes de j.

Qui = {}
for j in range(nc_nao,nc):
    Q = 0.0
    for l in coluna[j]:
        for soma in range(mais_mais_min[j],mais_mais_max[j]+1):
            if (esp[l,j,soma] != 0.0):
                Q = Q + (((dado[l,j,soma]-esp[l,j,soma])**2)/esp[l,j,soma])
    Qui[j] = Q
    #print "Qui[", j, "]=", Qui[j]
```

O valor:

$$Qui[2] = 245.0795378$$

é o valor do *Qui-Quadrado* calculado, referente à 3ª*coluna*.

Para finalizar, efectua-se o teste do *Qui-Quadrado*, com o intuito de ver se cada coluna j é dependente ou independente, relativamente à soma de todas as outras colunas. Assim, para cada coluna j começa-se por determinar os graus de liberdade correspondentes à mesma, que consiste no produto entre a diferença do maior valor para a soma dos valores presentes nas colunas diferentes de j e o menor, menos uma unidade, e a diferença entre o maior valor da coluna j e o menor, menos uma unidade.

Em seguida, compara-se o valor do *Qui-Quadrado* calculado e o tabelado, analogamente aos programas anteriores. Se o primeiro for maior ou igual que o segundo, então diz-se que a coluna j é dependente relativamente à soma das restantes colunas, caso contrário diz-se independente.

```
# Teste do Qui-Quadrado.
# gl[j] dá-nos os graus de liberdade entre a coluna j e as colunas diferentes de j.

gl = {}
auxiliar = 0
for j in range(nc_nao,nc):
    gl[j] = (mais_mais_max[j]-mais_mais_min[j]-1)*(coluna[j][valores[j]-1]-coluna[j][0]-1)
    #print "gl[", j, "] =", gl[j]
    #print "QQ[", gl[j], "][", 2,"] =", float(QQ[gl[j]][2])
    if (Qui[j] >= float(QQ[gl[j]][2])):
        print "As coluna", j, "contra a soma das colunas diferentes de", j, "representa um
caso de dependência"
    else:
        print "As coluna", j, "contra a soma das colunas diferentes de", j, "representa um
caso de independência"
```

Ao executar o programa descrito, nos 15 resultados obtidos apenas se podem constatar casos de dependência, ou seja, quando se considera uma coluna isoladamente, em relação à soma das restantes colunas, o resultado indica sempre dependência. Sob um ponto de vista genético, isto significa que um marcador apresenta sempre algum tipo de associação relativamente à soma dos restantes.

Capítulo 4

Conclusão

Após uma análise detalhada dos resultados obtidos no estudo efectuado foi possível obter uma resposta à questão central do presente trabalho. Note-se que o objectivo crucial deste trabalho era avaliar o grau de dependência envolvendo marcadores do tipo *Y-STR*.

Primeiramente realizou-se uma análise minuciosa dos resultados obtidos em cada um dos programas elaborados, chegando-se à conclusão que se pode extrair deste estudo.

No que diz respeito ao primeiro programa elaborado, *2colunas_Contiguidade.py*, o qual foi criado com o intuito de testar a verosimilhança da *Independência/Dependência* entre dois marcadores, supondo contíguidade entre os valores, analisando os resultados obtidos, constata-se uma notável maioria de casos de dependência.

Relativamente ao programa *3colunas_Contiguidade.py*, o qual foi criado nas mesmas condições que o programa anteriormente referido, mas considerando três marcadores em vez de dois, constata-se também um elevado número de casos de dependência entre os diferentes marcadores.

Analisando os resultados adquiridos no programa *2colunas* + *3colunas_Contiguidade.py*, o qual consiste na combinação dos dois programas anteriormente referidos, evidencia-se um número reduzido de casos que manifestam dependência entre três marcadores, mas independência entre os mesmos, quando considerados dois a dois.

No que refere ao programa *2colunas_Não Contiguidade.py*, o qual foi criado nas mesmas condições do programa *2colunas_Contiguidade.py*, à excepção da condição de contiguidade, que não foi imposta com o intuito de testar a influência da mesma nos resultados obtidos, observa-se também um elevado número de casos de dependência, pelo que se poderá afirmar que a não exigência da condição mencionada não provocou grandes alterações nos resultados adquiridos.

À semelhança do caso anterior, o programa *3colunas_Não Contiguidade.py* foi criado sob as mesmas condições do programa *3colunas_Contiguidade.py*, com excepção da condição de contiguidade de valores. Esta condição não foi imposta com a mesma finalidade do programa anterior, observando-se também um número reduzido de casos de independência. Analogamente ao caso anterior, poder-se-á dizer que a não imposição desta condição não alterou de forma considerável os resultados obtidos.

Centrando a análise nos resultados obtidos no programa *2colunas* + *3colunas_Não Contiguidade.py*, que consiste na combinação dos dois programas anteriormente mencionados, constata-se um número bastante reduzido de marcadores que manifestam dependência quando considerados três a três, mas independência quando os mesmos são considerados dois a dois. Analogamente aos programas anteriores, o presente programa em relação ao *2colunas* + *3colunas_Não Contiguidade.py* não apresentou grandes alterações nos resultados, pelo que se poderá afirmar que a condição de contiguidade não revelou grande influência nos resultados obtidos.

No que diz respeito ao programa *Ordenar por Probabilidades.py*, criado sob as mesmas condições do programa *2colunas_Contiguidade.py*, mas ordenando os dados por ordem decrescente de intervalos de probabillidade, depois graus de liberdade, e por fim valor do *Qui-Quadrado* determinado, não se verificou alteração dos resultados, confirmando-se um número elevado de casos de dependência entre dois marcadores.

Efectuando uma análise dos resultados obtidos no programa *Soma de n colunas.py*, elaborado com o objectivo de testar a verosimilhança da Dependência/Independência de n *marcadores*, impondo a soma dos valores nos mesmos, constatou-se um número bastante reduzido de casos de independência quando o referido programa é executado para dois marcadores, notando-se, ainda, uma dimiuição dos casos mencionados quando o mesmo é executado para três marcadores em vez de dois.

Relativamente ao programa *Soma das Diferenças nos Extremos.py*, criado sob as mesmas condições do programa anterior, sendo ainda imposta a soma das diferenças nos extremos, isto é, a soma mínima e a soma máxima possível nos respectivos marcadores, constata-se que os resultados obtidos não apresentam qualquer alteração relativamente ao programa anterior, manifestando-se um notável número de casos de dependência quando se executa o programa para dois ou para três marcadores.

Analisando os resultados obtidos no programa *Independência de 2 colunas e Sucessivamente.py*, criado com o objectivo de testar a verosimilhança da Independência de *n marcadores*, tendo em conta que estes o seriam quando considerados em subgrupos de *n-1 marcadores*, não é possível observar independência entre marcadores quando estes são considerados em dimensão igual ou superior a quatro, exigindo também uma independência entre os mesmos, quando considerados em subgrupos de dimensões inferiores.

Em relação ao programa *Um Contra Todos,py*, que permite estudar a hipótese de Dependência/Independência de uma coluna em relação à soma das restantes, apenas se podem observar casos de dependência.

Depois de uma análise pormenorizada dos resultados obtidos em cada um dos programas elaborados, é possível chegar a alguma conclusão. Os resultados do estudo evidenciam um considerável número de casos de dependência, independentemente da condição que possa ser imposta ao estudo. Assim, como resposta à questão fulcral do presente trabalho poderia dizer-se que o grau de dependência envolvido nos marcadores do tipo *Y-STR* é bastante elevado.

É de notar que este resultado confirma as hipóteses previamente avançadas pelos investigadores do IPATIMUP, uma vez que a população em estudo é bastante recente e a quebra de ligações é um fenómeno muito lento no tempo, isto é, não existe tempo suficiente entre os indivíduos que possibilite a ocorrência de fenómenos mutacionais representados pela independência dos marcadores.

Capítulo 5

Relevância do Resultado para o IPATIMUP

Na criação e utilização de bases de dados para fins forenses, quando se analisam vários *loci*, mesmo para um número relativamente modesto de marcadores e de alelos por marcador, o número esperado de combinações alélicas (ou haplótipos) é extremamente elevado. Sendo assim, o número de amostras/indivíduos que é necessário para obter estimativas de frequência fiáveis é extremamente alto. De facto, quando se estudam muitos loci e estes são muito polimórficos (isto é, têm muitos alelos comuns na população em estudo), o número de combinações possíveis é muito elevado. Se, por hipótese, se quisesse estudar 10 loci e cada um deles tivesse 5 alelos, o número de haplótipos possível seria de 5^{10}, ou seja de quase 10 milhões.

No entanto, caso a combinação dos alelos dos diferentes loci seja aleatória (isto é, quando se trata de marcadores genéticos que não estão associados) a probabilidade de um haplótipo pode ser obtida simplesmente multiplicando as frequências dos não alelos envolvidos. Dado que o número de alelos por loci é muito menor (a par de uma maior frequência alélica por locus), resulta que, para um mesmo nível de confiança, se tenha necessidade de analisar um número muito menor de amostras. Assim, com uma base de dados de tamanho relativamente modesto (da ordem de 10^2 indivíduos) é possível extrapolar com segurança probabilidades de constelações genéticas não observadas ou muito raras.

No campo da genética populacional, existem vários "programas" que permitem determinar a existência de associação entre pares de loci, mas que apresentam a limitação de apenas serem considerados 2 loci de cada vez (*linkage disequilibrium*). No entanto, quando se analisa simultaneamente um número de marcadores superior a 2, no caso de nenhum par de marcadores revelar associação, subsiste a dúvida se esta poderá mesmo assim existir quando consideramos simultaneamente 3 ou mais marcadores.

A partir dos resultados deste trabalho, é possível concluir que, apesar de existirem marcadores para os quais não se detecta associação com qualquer um dos restantes (quando utilizada a analise convencional de pares de loci) é possível verificar que todos eles estão envolvidos em algum tipo de associação quando analisado (com o programa desenvolvido neste trabalho) um número maior de marcadores simultaneamente (3 ou 4 loci).

A confirmação da existência de associação entre os 15 marcadores estudados comprova não ser legítimo extrapolar com segurança probabilidades haplotípicas a partir do produto das frequências dos não alelos envolvidos.

Deste modo, as bases de dados actualmente disponíveis, quer para a população portuguesa, quer para outras populações, não permitem estimativas fiáveis de frequências haplotípicas, devendo, por isso, o seu efectivo ser altamente incrementado.

Apêndice A

Tabela do Qui-Quadrado

gl	níveis de significância				
df	0.10	0.05	0.025	0.01	0.005
1	2.7055	3.8415	5.0239	6.6349	7.8794
2	4.6052	5.9915	7.3778	9.2104	10.5965
3	6.2514	7.8147	9.3484	11.3449	12.8381
4	7.7794	9.4877	11.1433	13.2767	14.8602
5	9.2363	11.0705	12.8325	15.0863	16.7496
6	10.6446	12.5916	14.4494	16.8119	18.5475
7	12.017	14.0671	16.0128	18.4753	20.2777
8	13.3616	15.5073	17.5345	20.0902	21.9549
9	14.6837	16.919	19.0228	21.666	23.5893
10	15.9872	18.307	20.4832	23.2093	25.1881
11	17.275	19.6752	21.92	24.725	26.7569
12	18.5493	21.0261	23.3367	26.217	28.2997
13	19.8119	22.362	24.7356	27.6882	29.8193
14	21.0641	23.6848	26.1189	29.1412	31.3194
15	22.3071	24.9958	27.4884	30.578	32.8015
16	23.5418	26.2962	28.8453	31.9999	34.2671
17	24.769	27.5871	30.191	33.4087	35.7184
18	25.9894	28.8693	31.5264	34.8052	37.1564

19	27.2036	30.1435	32.8523	36.1908	38.5821
20	28.412	31.4104	34.1696	37.5663	39.9969
21	29.6151	32.6706	35.4789	38.9322	41.4009
22	30.8133	33.9245	36.7807	40.2894	42.7957
23	32.0069	35.1725	38.0756	41.6383	44.1814
24	33.1962	36.415	39.3641	42.9798	45.5584
25	34.3816	37.6525	40.6465	44.314	46.928
26	35.5632	38.8851	41.9231	45.6416	48.2898
27	36.7412	40.1133	43.1945	46.9628	49.645
28	37.9159	41.3372	44.4608	48.2782	50.9936
29	39.0875	42.5569	45.7223	49.5878	52.3355
30	40.256	43.773	46.9792	50.8922	53.6719
31	41.4217	44.9853	48.2319	52.1914	55.0025
32	42.5847	46.1942	49.4804	53.4857	56.328
33	43.7452	47.3999	50.7251	54.7754	57.6483
34	44.9032	48.6024	51.966	56.0609	58.9637
35	46.0588	49.8018	53.2033	57.342	60.2746
36	47.2122	50.9985	54.4373	58.6192	61.5811
37	48.3634	52.1923	55.668	59.8926	62.8832
38	49.5126	53.3835	56.8955	61.162	64.1812
39	50.6598	54.5722	58.1201	62.4281	65.4753
40	51.805	55.7585	59.3417	63.6908	66.766
41	52.9485	56.9424	60.5606	64.95	68.0526
42	54.0902	58.124	61.7767	66.2063	69.336
43	55.2302	59.3035	62.9903	67.4593	70.6157
44	56.3685	60.4809	64.2014	68.7096	71.8923
45	57.5053	61.6562	65.4101	69.9569	73.166
46	58.6405	62.8296	66.6165	71.2015	74.4367
47	59.7743	64.0011	67.8206	72.4432	75.7039
48	60.9066	65.1708	69.0226	73.6826	76.9689
49	62.0375	66.3387	70.2224	74.9194	78.2306
50	63.1671	67.5048	71.4202	76.1538	79.4898
51	64.2954	68.6693	72.616	77.386	80.7465
52	65.4224	69.8322	73.8099	78.6156	82.0006
53	66.5482	70.9934	75.0019	79.8434	83.2525

54	67.6728	72.1532	76.1921	81.0688	84.5018
55	68.7962	73.3115	77.3804	82.292	85.7491
56	69.9185	74.4683	78.5671	83.5136	86.994
57	71.0397	75.6237	79.7522	84.7327	88.2366
58	72.1598	76.7778	80.9356	85.9501	89.477
59	73.2789	77.9305	82.1174	87.1658	90.7153
60	74.397	79.082	83.2977	88.3794	91.9518
61	75.5141	80.2321	84.4764	89.5912	93.1862
62	76.6302	81.381	85.6537	90.8015	94.4185
63	77.7454	82.5287	86.8296	92.0099	95.6492
64	78.8597	83.6752	88.004	93.2167	96.8779
65	79.973	84.8206	89.1772	94.422	98.1049
66	81.0855	85.9649	90.3488	95.6256	99.3303
67	82.1971	87.108	91.5193	96.8277	100.5538
68	83.3079	88.2502	92.6885	98.0283	101.7757
69	84.4179	89.3912	93.8565	99.2274	102.9961
70	85.527	90.5313	95.0231	100.4251	104.2148
71	86.6354	91.6703	96.1887	101.6214	105.4323
72	87.7431	92.8083	97.353	102.8163	106.6473
73	88.8499	93.9453	98.5162	104.0098	107.8619
74	89.9561	95.0815	99.6784	105.2019	109.0742
75	91.0615	96.2167	100.8393	106.3929	110.2854
76	92.1662	97.351	101.9992	107.5824	111.4954
77	93.2702	98.4844	103.1581	108.7709	112.7037
78	94.3735	99.617	104.3159	109.9582	113.9107
79	95.4762	100.7486	105.4727	111.144	115.1163
80	96.5782	101.8795	106.6285	112.3288	116.3209
81	97.6796	103.0095	107.7834	113.5123	117.524
82	98.7803	104.1387	108.9373	114.6948	118.7261
83	99.8805	105.2672	110.0902	115.8762	119.927
84	100.98	106.3949	111.2422	117.0566	121.1262
85	102.0789	107.5217	112.3933	118.2356	122.3244
86	103.1773	108.6479	113.5436	119.4137	123.5218
87	104.275	109.7733	114.6929	120.5909	124.7176
88	105.3723	110.898	115.8415	121.7672	125.9123

89	106.4689	112.022	116.989	122.9422	127.106
90	107.565	113.1452	118.1359	124.1162	128.2987
91	108.6606	114.2679	119.282	125.2893	129.4902
92	109.7556	115.3898	120.427	126.4616	130.6812
93	110.8501	116.511	121.5714	127.633	131.8705
94	111.9442	117.6317	122.7152	128.8032	133.0589
95	113.0377	118.7516	123.858	129.9725	134.2466
96	114.1307	119.8709	125.0001	131.1411	135.4327
97	115.2232	120.9897	126.1414	132.3089	136.6188
98	116.3153	122.1077	127.2821	133.4756	137.803
99	117.4069	123.2252	128.4219	134.6415	138.9869
100	118.498	124.3421	129.5613	135.8069	140.1697

Apêndice B

Tabela YPortugal_Total2.csv

marcadores

Reg.	IDNr.	Y19	Y389I	Y389II	Y390	Y391	Y392	Y393	Y437	Y438	Y439	A10	Y360	Y361	Y635	H4
B	2445	13	13	17	24	10	11	13	14	10	13	16	9	12	22	28
VC	3700	16	12	17	22	9	12	13	16	10	13	14	10	13	21	28
VC	3495	13	13	18	24	10	11	12	14	10	12	14	10	13	24	28
VC	3692	13	14	19	24	10	11	12	14	10	12	14	10	13	23	28
B	528	13	12	17	24	10	11	13	14	10	10	15	10	12	21	27
BN	756	13	13	17	24	10	11	12	14	10	12	15	9	12	23	28
P	1174	13	13	18	23	10	11	13	14	10	12	15	11	11	22	28
P	1238	13	12	17	24	10	11	13	14	10	10	15	10	13	21	27
P	1290	13	13	17	24	10	11	12	14	10	12	15	9	12	23	28
P	1942	14	13	18	24	10	11	13	14	10	12	15	11	12	21	29
P	2088	13	13	17	25	10	11	13	14	10	13	14	9	12	21	27
VR	3260	14	13	17	24	10	11	13	14	10	12	14	10	11	20	28
B	884	14	14	16	24	9	11	13	14	10	10	15	10	13	21	28
B	2094	13	13	16	24	9	11	13	14	10	10	14	11	13	21	28
B	2999	13	14	16	24	9	11	13	14	10	10	14	11	12	21	28
BN	2408	13	14	16	24	9	11	13	14	10	11	14	11	13	21	27
P	1097	13	13	16	25	9	11	13	14	10	10	14	11	12	22	28
P	1171	13	13	16	24	9	11	13	14	10	11	14	11	14	22	28

P	1270	13	13	16	24	9	11	13	14	10	11	14	11	12	21	27
P	1356	13	14	16	24	9	11	13	14	10	10	14	11	13	21	28
P	1547	13	14	16	24	9	11	13	14	10	10	14	11	13	21	29
P	1561	13	14	16	24	9	11	13	14	10	11	14	11	13	21	28
P	1601	13	14	16	24	9	11	13	14	10	10	14	11	14	21	28
P	2184	13	13	16	24	9	11	13	14	10	10	14	11	13	21	28
VC	3101	13	14	16	23	9	12	13	14	10	10	14	11	13	20	28
VC	3517	13	13	16	25	9	11	13	14	10	11	14	11	14	21	28
VC	3522	13	13	16	24	9	11	13	14	10	10	14	11	13	21	28
VC	3679	13	14	16	24	9	11	13	14	10	10	14	11	12	21	28
VR	3232	13	13	16	24	9	11	13	14	10	10	14	11	13	21	27
B	1873	15	12	17	23	10	11	13	14	10	12	15	10	12	22	27
BN	2562	13	13	17	25	10	11	14	14	10	13	13	10	12	21	28
P	1806	15	13	17	24	10	11	12	14	10	11	14	10	14	22	26
P	1856	13	12	17	24	10	11	13	14	10	13	14	10	11	23	27
B	1081	15	12	16	21	10	11	14	16	10	12	13	10	11	21	27
B	2262	15	12	16	23	9	11	14	16	10	11	13	11	11	21	27
B	2675	15	12	17	22	10	11	14	16	10	11	13	9	11	21	27
B	3329	16	13	17	22	11	10	13	16	10	13	14	11	12	21	27
BN	1962	16	12	17	22	11	11	13	16	10	11	14	10	11	20	27
P	1233	15	12	16	21	10	11	15	15	10	11	15	11	11	21	27
P	1298	15	12	16	21	10	11	14	16	11	11	13	10	11	23	27
P	1502	15	12	17	22	10	11	14	16	10	12	15	11	10	21	27
P	1586	16	12	16	22	10	11	14	16	11	11	13	11	11	20	29
P	2015	15	12	18	23	10	12	14	16	9	12	14	10	12	21	28
P	2029	15	12	16	23	10	11	14	16	10	12	14	10	11	20	27
P	2090	15	13	17	22	11	11	13	16	10	12	14	10	12	21	28
VC	2192	15	12	17	22	10	11	14	15	10	13	14	11	12	21	27
VC	2207	15	12	17	22	11	11	13	15	9	10	14	10	11	21	27
VC	3693	16	12	16	22	10	12	14	16	11	11	13	11	10	21	27
VR	3677	15	12	17	23	10	11	14	16	10	12	14	10	11	20	27
B	149	16	12	17	24	10	12	14	15	10	10	16	11	12	19	28
B	1766	15	13	17	23	10	12	15	14	10	11	14	11	12	21	28
B	2162	14	12	16	23	10	11	13	16	10	11	15	10	12	21	28
B	2188	15	14	18	22	10	12	15	14	10	11	14	11	12	21	27

B	2235	15	13	17	23	11	12	15	15	10	11	15	10	12	19	27
B	2577	14	12	16	23	11	11	13	16	10	11	15	10	12	22	27
B	2654	14	12	17	22	10	11	13	16	10	12	14	11	12	21	28
B	3159	14	12	16	22	10	11	13	16	10	12	14	10	12	21	29
BN	687	14	12	16	23	10	11	13	16	10	11	14	10	12	22	27
BN	2106	14	12	16	24	11	11	13	16	10	11	15	10	11	22	27
BN	2256	14	12	16	22	10	11	12	16	10	12	15	10	12	24	27
P	1443	16	13	16	24	11	12	14	15	10	11	15	11	12	20	28
P	1642	15	13	17	24	10	12	15	15	10	11	15	11	12	19	28
P	1834	14	12	16	22	10	11	13	16	10	11	15	11	12	24	28
P	1932	15	13	17	23	10	12	15	14	10	11	14	10	12	23	27
VC	2967	15	14	17	24	10	12	15	15	10	11	15	11	12	20	28
VC	3512	13	13	17	24	10	11	13	14	10	12	15	9	12	22	27
VC	3515	14	12	16	23	10	11	13	16	10	11	14	10	13	21	27
VC	3684	15	13	18	22	10	12	14	15	10	11	15	11	12	21	27
VC	3687	15	12	18	23	10	12	15	15	10	13	18	11	13	19	28
VR	851	16	13	17	23	11	11	13	14	10	10	13	10	11	21	27
VR	1070	14	12	17	23	10	11	13	16	10	12	15	10	13	22	27
VR	3457	14	12	16	22	10	11	13	16	10	11	15	10	12	21	29
B	2121	16	14	17	23	11	11	13	15	10	11	17	11	11	21	29
BN	2593	16	13	15	23	10	11	13	15	10	11	16	10	11	22	27
P	1846	17	13	15	23	10	11	13	15	10	11	16	10	11	22	27
VR	1189	15	13	15	23	11	11	13	15	10	11	14	10	12	21	29
VR	2845	15	13	15	23	11	11	13	15	10	11	16	10	12	21	28
B	1014	14	13	17	24	12	11	12	14	10	10	16	10	11	21	27
B	2979	14	13	17	23	10	12	12	14	10	11	17	11	11	20	28
P	1499	14	13	17	23	10	11	12	14	10	12	15	11	11	21	27
P	1668	14	14	17	23	10	11	12	14	10	11	16	11	11	20	27
P	1948	14	14	16	23	10	11	12	14	10	11	14	11	13	20	27
P	2047	14	13	18	25	10	11	12	14	10	11	15	11	11	21	27
VC	3686	14	13	17	23	10	11	12	14	10	11	14	11	11	20	27
VC	3689	15	12	17	23	10	11	12	14	10	11	15	10	12	20	27
BN	1706	15	13	17	23	11	11	12	15	9	13	16	10	13	25	27
P	1200	15	12	16	24	10	11	13	16	9	11	14	11	10	21	27
P	1206	13	14	16	24	11	11	12	16	9	13	14	11	13	20	30

P	1376	15	13	16	23	10	11	12	15	9	11	14	11	10	21	27
P	1416	14	14	18	23	10	11	13	14	9	12	14	9	13	22	27
P	1417	15	14	17	23	11	11	12	16	9	11	15	10	13	22	26
P	1882	15	12	16	24	10	11	12	16	9	12	14	11	10	21	27
P	1914	15	13	16	22	11	11	12	15	9	11	15	10	13	24	28
P	1998	15	13	16	23	9	11	13	14	9	11	14	10	13	24	28
P	2153	14	13	16	23	10	11	12	14	9	10	15	10	12	21	27
P	2173	15	12	17	23	10	11	12	15	9	10	15	10	12	23	27
VC	830	14	14	17	23	10	11	14	15	9	12	15	9	13	23	28
VC	1919	15	13	16	23	10	11	12	14	9	12	15	10	14	21	28
VC	3493	14	13	16	22	9	11	12	15	9	11	14	11	12	23	27
VC	3513	15	13	16	23	9	11	12	14	9	12	15	10	13	22	28
VC	3514	14	13	16	24	10	11	12	15	9	11	14	11	12	21	28
VC	3523	15	13	16	23	9	11	13	14	9	11	15	10	13	22	28
VC	3680	15	14	17	23	11	11	12	16	12	11	15	10	13	23	28
VC	3683	14	12	16	23	10	11	12	14	9	11	15	11	15	22	29
VC	3703	15	13	17	25	10	11	12	16	9	11	15	11	10	22	27
VR	699	14	13	16	23	10	11	12	15	9	11	16	11	14	22	28
VR	1136	14	13	17	24	10	11	12	15	9	12	15	11	12	23	28
VR	3466	14	14	18	23	10	14	13	15	9	12	15	10	13	21	27
B	892	15	13	16	24	10	13	13	14	9	12	14	10	11	21	27
B	1949	16	14	17	23	10	13	13	14	10	10	15	10	11	20	28
P	1447	14	13	16	23	10	13	13	14	9	11	14	11	11	21	27
P	1459	14	13	16	23	10	13	13	14	9	11	14	11	11	21	27
P	2177	15	12	15	24	10	14	13	14	9	11	15	11	11	21	27
VC	3467	15	12	16	24	10	13	13	14	9	11	14	10	12	23	27
VR	925	15	13	17	23	10	13	13	14	9	11	17	10	11	21	27
VR	2164	15	12	15	24	10	14	13	14	9	11	15	10	11	21	27
VR	3462	15	14	17	23	10	13	13	14	10	11	16	10	11	21	27
P	2058	14	14	16	23	10	15	12	16	10	11	14	10	11	25	26
B	2000	16	13	19	24	10	11	13	14	10	10	17	11	11	24	29
B	2250	15	13	18	24	11	13	13	16	11	12	17	11	11	24	28
P	1390	16	14	16	24	10	11	14	14	11	10	16	12	11	23	29
P	2014	16	13	18	24	10	11	14	14	11	10	16	11	11	23	28
VC	3519	15	13	18	25	11	11	13	14	11	10	16	11	11	23	29

VR	2042	16	13	18	25	11	11	13	14	11	10	16	12	11	23	27
VR	2965	15	13	18	25	11	11	13	14	11	10	16	11	11	23	29
B	65	15	12	17	25	11	11	13	15	12	13	15	11	12	24	30
B	130	14	13	16	24	10	14	13	15	12	11	15	10	12	23	28
B	308	14	13	16	23	11	13	13	14	12	12	15	10	12	23	27
B	414	15	13	16	24	10	13	13	15	12	12	15	11	12	23	28
B	550	14	12	16	24	11	14	13	15	12	12	15	11	12	23	27
B	655	15	13	17	24	11	13	14	15	12	12	14	10	12	23	28
B	896	14	13	16	23	10	14	13	15	12	11	16	11	12	23	28
B	976	14	13	15	24	11	13	12	15	12	13	15	10	12	23	28
B	1016	15	13	16	24	10	12	14	15	12	11	15	11	12	23	28
B	1549	14	14	16	24	10	13	13	14	12	11	14	11	12	23	27
B	1664	14	13	16	23	11	14	13	14	12	10	14	10	12	23	28
B	1675	14	13	16	24	11	14	13	14	12	11	14	11	12	23	27
B	1719	13	14	16	23	11	13	13	15	12	11	16	11	12	23	28
B	1744	14	14	16	23	11	13	13	15	12	12	12	11	12	23	28
B	1787	14	13	16	24	11	13	12	15	12	12	15	10	11	24	29
B	1819	15	13	16	24	11	13	13	14	12	11	15	11	10	23	27
B	2007	15	13	17	24	11	13	13	14	12	12	14	11	12	23	28
B	2208	14	13	17	25	11	13	13	15	12	12	15	11	12	23	28
B	2442	14	13	16	24	11	13	13	15	12	12	15	10	12	23	28
B	2583	14	14	16	24	11	13	13	15	12	12	15	11	12	23	28
B	2810	14	12	16	24	11	14	13	15	12	13	15	11	12	23	28
B	3088	14	12	16	24	11	13	13	15	12	12	15	11	12	23	28
B	3098	14	13	16	24	11	13	13	15	12	12	15	10	12	23	28
B	3285	14	13	17	24	11	13	13	15	12	12	15	11	12	23	29
B	3323	14	14	16	23	11	13	14	14	12	13	14	11	12	23	29
BN	554	14	13	16	24	11	13	13	15	12	13	15	10	12	24	28
BN	684	14	13	16	23	11	13	13	14	12	12	14	10	12	23	27
BN	982	15	13	16	24	11	13	13	15	12	13	14	11	13	23	27
BN	984	15	13	16	25	10	14	13	15	12	12	14	10	12	23	28
BN	1211	14	12	16	24	11	13	13	15	12	13	16	10	12	23	28
BN	1276	14	14	17	24	11	13	13	15	12	12	15	11	12	23	28
BN	1316	14	13	16	24	11	13	13	14	12	12	15	10	12	23	28
BN	1424	14	14	16	25	10	13	13	15	13	13	16	11	12	23	28

BN	1435	14	12	16	25	11	13	13	15	12	12	14	11	12	24	28
BN	1437	14	13	16	23	10	13	12	15	12	12	14	10	12	23	27
BN	1823	14	12	16	25	10	12	13	15	12	11	15	10	12	24	28
BN	1827	14	13	16	25	10	13	14	16	12	13	15	11	12	23	28
BN	1905	14	13	17	24	12	13	13	15	12	12	16	11	12	23	28
BN	2209	14	12	16	24	11	13	13	15	10	13	15	10	12	24	27
BN	2595	14	13	16	23	10	13	13	14	12	14	15	11	12	23	27
BN	2670	16	13	16	24	11	13	13	15	12	12	15	10	12	23	27
P	1207	14	14	15	24	10	14	12	15	12	13	15	10	11	24	28
P	1285	14	14	17	24	11	13	13	15	12	12	15	11	12	23	28
P	1291	14	13	16	23	11	13	13	15	12	13	15	10	12	24	27
P	1313	14	13	15	25	11	13	13	15	12	11	15	11	12	23	28
P	1343	14	13	16	24	11	13	13	15	10	12	15	10	12	23	28
P	1353	14	13	15	23	10	13	13	15	12	11	15	11	12	23	28
P	1355	15	14	16	24	11	13	13	15	12	11	16	11	12	23	27
P	1381	15	13	16	24	11	13	12	14	12	12	15	12	11	23	28
P	1382	14	14	16	25	10	13	13	15	12	12	15	11	12	23	27
P	1414	14	13	16	23	11	13	13	15	13	12	14	11	11	24	28
P	1415	14	12	16	24	10	13	13	14	12	13	15	11	12	23	27
P	1429	14	14	16	24	11	13	13	15	12	11	15	10	12	26	28
P	1457	14	13	17	24	11	13	13	14	12	12	15	10	12	23	27
P	1458	15	13	16	25	10	13	12	15	13	11	14	10	11	23	29
P	1465	14	13	16	24	11	13	13	15	12	13	15	11	12	24	27
P	1484	15	13	16	24	11	13	13	15	12	13	16	11	12	23	28
P	1487	15	14	17	24	10	15	13	15	12	13	15	11	12	23	27
P	1491	14	12	16	25	11	13	13	15	12	13	15	11	12	23	28
P	1492	14	13	16	23	11	13	13	14	12	11	16	11	12	23	28
P	1505	14	14	16	24	11	13	13	15	11	13	13	10	12	24	27
P	1565	14	13	16	24	11	13	13	15	12	11	15	11	12	25	28
P	1566	14	14	16	24	11	13	13	15	12	12	15	11	12	23	28
P	1569	14	13	16	24	11	13	13	14	11	11	14	11	12	23	28
P	1573	13	15	16	25	11	13	13	15	12	12	15	11	12	23	27
P	1576	14	13	16	24	11	13	14	15	12	11	15	10	13	23	28
P	1580	14	14	16	24	11	13	13	15	12	12	15	10	12	23	28
P	1584	14	13	16	24	10	13	13	15	12	12	15	12	12	23	28

P	1588	14	13	16	24	11	13	13	15	12	12	14	10	12	23	29
P	1602	14	13	16	24	10	13	13	15	12	12	14	11	12	24	28
P	1609	14	13	16	25	12	13	13	14	12	12	15	11	12	23	28
P	1622	14	13	16	24	10	13	13	15	12	12	15	11	12	24	28
P	1644	14	14	16	24	11	13	13	14	12	12	17	11	12	23	27
P	1659	14	14	16	24	11	13	13	15	12	12	15	11	12	24	27
P	1679	14	14	17	24	11	13	13	15	12	12	15	10	13	23	28
P	1694	14	13	16	25	10	14	13	15	12	13	14	10	12	23	28
P	1698	14	13	16	23	11	13	13	16	12	11	15	10	12	23	28
P	1757	14	13	16	24	10	13	13	14	12	12	14	11	12	23	29
P	1770	15	13	16	24	10	13	13	15	12	12	15	11	12	23	29
P	1773	15	14	16	24	10	14	12	15	12	11	15	11	11	23	28
P	1782	15	13	16	24	10	13	13	15	12	12	16	11	12	23	29
P	1839	14	13	16	23	11	13	13	15	12	12	15	10	12	23	28
P	1845	14	14	16	24	11	13	13	15	12	12	15	10	12	23	28
P	1875	14	13	16	23	11	13	13	15	12	11	15	10	12	23	27
P	1936	14	14	16	24	11	13	13	15	12	11	14	10	12	23	28
P	1963	14	12	16	24	10	13	13	15	12	12	15	10	12	23	28
P	1969	14	13	15	23	10	13	13	15	12	11	14	11	12	23	28
P	1978	14	13	16	25	11	13	13	15	12	12	15	10	12	23	28
P	1983	14	13	16	24	10	13	12	15	12	11	16	11	11	23	28
P	1999	14	13	16	24	11	13	13	15	12	11	15	10	12	25	28
P	2023	14	14	16	25	10	14	13	15	12	13	14	10	12	23	29
P	2026	14	13	16	24	11	13	13	15	12	9	15	11	12	23	27
P	2030	14	11	15	24	11	13	13	15	12	13	14	11	13	24	28
P	2048	14	13	17	24	10	13	13	15	12	11	16	11	12	23	28
P	2060	14	13	16	24	10	13	13	14	12	11	15	10	12	26	28
P	2063	14	13	15	25	10	13	13	15	12	11	15	10	12	23	28
P	2082	14	13	16	24	11	13	14	15	12	11	14	10	12	23	28
P	2099	15	11	15	24	10	13	14	15	12	12	15	11	12	23	28
P	2100	14	13	15	26	11	13	12	15	12	12	14	10	11	23	28
P	2107	14	13	16	24	11	13	13	14	12	12	15	10	12	23	28
P	2120	14	13	16	24	10	13	13	14	12	12	16	11	12	25	28
P	2127	14	13	16	24	11	13	12	15	12	12	15	10	11	24	29
P	2128	15	13	16	22	11	13	13	15	12	13	15	11	12	23	28

P	2147	14	13	17	23	10	13	13	15	12	11	15	10	14	23	29
P	2148	14	14	16	24	11	13	14	15	12	12	15	10	12	23	27
P	2163	14	12	16	23	10	13	13	15	12	12	15	10	12	23	29
P	2174	14	13	17	25	11	14	13	14	12	12	15	12	12	23	28
P	2181	14	13	16	24	11	13	13	14	12	11	15	11	12	23	27
VC	459	14	13	16	24	10	13	13	15	12	12	15	11	12	24	29
VC	977	14	13	16	24	11	13	13	14	9	12	15	10	12	24	28
VC	1667	14	13	16	23	10	13	13	15	12	11	15	10	14	23	29
VC	1918	14	13	16	25	11	13	13	15	12	12	14	10	13	23	28
VC	1947	14	13	16	24	11	13	13	14	12	11	15	11	12	23	28
VC	2677	14	13	16	24	10	13	13	15	12	12	14	10	12	23	28
VC	2734	14	13	15	23	10	13	13	15	12	11	15	11	12	23	28
VC	3158	14	13	16	23	11	13	13	14	12	11	14	10	12	23	28
VC	3283	14	13	16	24	12	13	13	15	12	11	14	11	12	23	28
VC	3284	14	13	16	24	13	13	13	14	13	12	15	11	12	23	27
VC	3346	13	14	16	25	10	13	13	15	12	12	15	11	12	23	27
VC	3490	14	13	16	24	11	13	13	15	12	12	14	11	11	23	28
VC	3491	15	13	16	24	11	13	13	15	12	13	16	10	11	23	30
VC	3492	15	13	16	24	11	13	13	15	12	12	15	11	12	23	28
VC	3496	14	13	16	25	10	13	13	15	12	13	14	10	11	23	28
VC	3497	14	13	16	24	10	13	13	15	12	11	15	10	13	23	28
VC	3516	14	13	16	24	10	13	13	15	12	11	15	12	13	23	28
VC	3518	15	13	16	24	11	13	14	16	12	12	14	10	12	23	27
VC	3681	14	13	16	23	11	13	13	15	12	12	15	12	13	23	28
VC	3682	15	14	15	24	10	13	13	15	12	11	15	11	11	23	28
VC	3685	14	13	16	23	11	13	13	15	12	11	15	10	12	23	30
VC	3691	14	13	16	24	10	13	13	15	12	13	15	11	11	23	28
VC	3694	14	13	16	24	11	13	13	14	12	12	15	11	12	23	27
VC	3695	14	13	17	24	12	13	13	15	12	12	15	11	12	24	27
VC	3696	15	13	16	23	11	14	13	15	12	12	14	11	12	23	29
VC	3697	15	13	16	24	10	13	13	15	12	12	15	11	12	23	28
VC	3698	14	13	16	23	11	13	13	14	12	12	15	9	12	23	27
VC	3699	14	13	16	24	10	13	12	14	13	11	14	11	12	23	28
VC	3701	14	13	16	24	11	13	13	15	12	13	15	13	12	23	28
VC	3702	14	13	16	25	11	13	13	15	12	13	15	10	12	24	28

VR	41	14	14	16	24	11	13	13	15	12	12	15	11	13	23	29
VR	98	14	13	15	24	10	13	13	15	13	11	15	10	12	23	28
VR	527	14	13	16	24	10	13	13	15	12	12	14	10	12	23	28
VR	875	14	14	16	25	11	14	13	15	12	12	14	10	12	23	29
VR	926	14	13	17	24	11	13	12	14	12	12	15	11	12	23	28
VR	956	14	14	16	23	11	13	13	14	12	11	14	10	12	23	28
VR	1724	14	13	17	23	12	13	12	15	12	13	15	12	11	24	29
VR	1917	14	13	16	23	11	13	13	15	12	13	15	11	12	24	27
VR	1957	14	13	16	23	11	13	13	15	12	12	14	10	12	23	27
VR	2112	14	13	16	24	10	13	14	16	12	13	15	11	13	23	28
VR	2987	14	13	16	24	11	13	13	15	12	12	16	11	12	24	27
VR	3281	14	12	16	24	10	13	13	15	13	11	15	10	12	23	28
VR	3459	14	13	15	25	10	13	13	15	12	12	15	11	12	23	27
VR	3461	14	13	17	24	10	14	13	15	12	12	15	9	14	23	27
VR	3463	14	15	16	24	11	13	13	15	12	11	15	10	12	24	28
VR	3665	14	13	16	24	10	13	13	16	12	12	15	10	12	23	28
VR	3672	14	13	16	24	10	13	13	15	12	12	15	10	13	23	27
VR	3673	14	14	16	25	10	14	13	15	12	13	14	10	12	23	29
VR	3674	14	13	16	24	11	13	13	15	12	12	15	11	12	23	28
VR	3675	14	13	16	23	10	13	13	15	12	12	15	11	12	23	28
VR	3676	14	13	16	24	10	13	13	15	12	12	14	11	13	23	28
VR	3678	14	14	16	25	10	14	13	14	12	12	14	10	12	23	29
P	1344	14	14	16	24	11	13	13	15	12	12	15	12	13	23	28
P	1818	15	13	16	23	11	13	12	15	12	12	15	11	12	24	28
P	1926	14	13	16	24	11	13	13	15	12	11	16	10	12	23	28
VR	3464	14	13	17	24	10	13	13	15	12	12	15	11	13	23	28
BN	87	14	13	16	24	11	13	13	15	13	13	14	11	12	23	28
BN	3405	14	12	16	23	11	11	13	16	10	11	15	10	11	22	27
BN	3529	14	13	16	24	11	13	13	15	12	12	15	11	12	23	28
BN	3555	14	14	16	24	9	11	13	14	10	10	14	10	13	22	27
BN	3654	15	12	16	22	10	11	13	16	10	12	15	10	13	21	27
BN	3841	14	13	16	24	11	13	13	14	12	12	15	10	12	23	28
BN	3898	15	13	16	24	11	13	13	15	12	13	15	11	12	23	28
BN	3918	14	14	16	25	11	13	12	15	12	13	14	10	11	24	29
BN	3921	14	14	16	24	11	13	13	14	9	12	15	10	12	24	27

BN	3969	17	13	16	27	10	11	13	14	11	10	15	11	11	24	28
BN	4017	14	12	16	25	11	13	13	14	12	12	15	11	14	23	27
BN	4027	14	13	19	22	10	11	12	14	10	12	15	11	11	20	27
BN	4095	13	13	17	24	10	11	13	14	10	12	15	9	12	22	28
BN	4147	14	12	16	23	10	11	12	14	9	11	14	11	14	23	27
BN	4151	14	13	16	23	10	14	13	15	13	11	16	11	12	23	28
BN	4165	14	13	16	24	10	11	12	15	9	11	15	11	13	21	27
BN	4313	14	12	15	24	11	13	13	15	12	13	14	11	12	23	28
BN	4344	14	13	16	27	10	13	13	15	12	12	15	11	12	23	28
BN	4393	14	13	17	23	10	11	12	14	10	11	15	10	12	20	27
BN	4554	14	13	17	24	11	13	13	15	12	12	15	10	12	23	28
P	1394	14	12	16	22	10	11	13	16	10	11	16	10	12	21	27
P	1562	14	13	16	25	10	13	14	15	12	11	15	10	10	23	28
B	256	14	13	16	24	11	13	13	15	12	13	14	10	12	24	28
B	1262	14	13	17	24	11	14	13	15	12	12	15	10	12	23	27
B	1953	14	13	17	23	8	11	13	14	10	10	14	11	13	21	28
VC	3521	14	10	16	24	11	14	13	15	12	13	15	10	12	23	28
VC	3688	15	13	18	24	10	11	12	14	10	12	14	10	13	21	26
Co	2796	14	13	16	24	11	13	13	15	11	11	15	11	12	23	29
Av	482	14	13	15	24	11	13	13	15	12	12	15	11	12	23	28
Av	649	14	12	15	24	10	13	14	15	12	11	15	11	12	23	27
Av	722	14	14	16	24	11	13	13	15	11	12	16	13	12	23	27
Av	745	15	13	17	24	10	13	13	14	12	12	16	11	12	25	29
Av	801	14	13	16	24	10	13	13	15	12	11	15	11	12	23	28
Av	917	14	14	16	23	11	13	13	15	12	12	15	10	12	23	27
Av	941	15	13	15	23	11	13	13	15	13	11	15	11	12	23	27
Av	964	13	13	16	24	11	14	13	14	12	12	15	11	12	23	27
Av	1006	14	13	16	23	11	14	12	14	13	12	16	11	11	23	28
Av	1046	14	12	17	24	11	13	13	15	10	13	15	10	13	23	28
Av	1116	15	13	16	23	11	13	14	15	12	12	15	11	11	23	28
Av	1154	14	13	16	24	11	13	13	15	12	12	15	10	13	23	28
Av	1241	14	13	16	24	11	13	12	15	13	13	15	11	11	24	29
Av	1454	14	13	16	24	11	13	13	15	12	12	14	10	12	23	28
Av	1493	14	13	17	24	11	13	13	15	12	12	15	10	12	23	28
Av	1548	14	13	16	23	11	13	13	15	12	12	14	11	12	23	27

Av	1621	14	13	18	24	10	13	13	15	12	12	15	11	12	23	28
Av	1688	15	13	16	24	10	13	13	15	12	12	15	11	12	23	29
Av	1765	14	13	17	24	10	13	13	15	12	12	14	10	12	23	28
Av	1906	14	13	16	23	10	13	13	15	12	13	15	11	13	23	28
Av	2239	14	14	16	24	11	13	13	15	12	12	15	11	12	23	27
Av	2974	15	14	16	23	10	13	13	15	12	12	15	10	12	24	28
Be	382	14	13	16	23	11	13	13	15	13	12	14	11	12	23	28
Be	814	14	14	17	23	11	13	14	15	12	12	14	11	12	23	28
Be	4789	15	13	16	24	11	13	13	15	12	12	15	11	12	23	28
CB	457	14	14	16	24	11	13	13	14	12	11	15	11	12	23	27
CB	2622	14	13	16	24	11	13	13	15	12	12	15	11	12	24	28
CB	2623	14	13	16	24	11	13	12	14	13	12	15	11	12	23	28
CB	2624	15	13	16	24	10	13	13	15	12	12	15	11	12	23	28
CB	2626	14	13	16	25	11	13	13	15	12	12	15	10	13	23	28
CB	2627	14	14	16	24	11	13	13	15	12	12	15	11	12	23	29
CB	2628	14	13	16	24	10	13	13	15	12	12	15	10	12	24	28
CB	2629	15	13	16	26	11	13	13	16	12	12	15	10	12	23	28
CB	2630	14	13	16	25	11	13	13	15	12	12	15	10	12	23	28
CB	2631	14	14	16	24	11	13	13	15	12	12	15	10	12	23	28
CB	2632	14	14	16	24	10	13	13	14	12	11	15	10	12	23	27
CB	2634	14	13	16	24	11	14	13	15	12	12	15	10	12	23	27
CB	2635	14	13	16	24	11	14	13	15	12	12	16	11	12	23	27
CB	2636	15	13	17	24	11	13	14	15	12	12	14	10	12	23	28
CB	2637	14	13	16	24	11	13	13	14	12	12	15	11	12	23	28
CB	2638	14	14	16	24	11	13	13	15	12	12	16	11	12	23	28
CB	2639	15	13	17	23	10	13	13	15	12	12	15	10	12	23	28
CB	2640	14	13	16	24	11	13	12	14	13	11	15	11	12	23	28
CB	2643	14	13	16	24	11	13	13	15	12	12	15	10	12	23	28
Co	2768	14	12	16	24	10	13	13	14	12	13	15	10	12	23	27
Co	2769	14	13	16	24	12	13	13	15	12	12	15	11	12	23	28
Co	2770	14	13	16	24	10	13	13	15	12	12	14	10	12	23	29
Co	2771	15	12	14	23	11	13	13	14	12	10	15	10	12	24	28
Co	2772	14	13	16	24	12	13	13	15	12	12	16	10	12	23	28
Co	2773	15	13	16	24	11	13	13	15	12	11	15	11	12	23	29
Co	2790	14	14	16	24	10	13	13	15	12	12	14	11	12	23	27

Co	2795	14	13	16	24	10	13	13	15	12	12	14	11	12	23	28
Co	3077	14	13	16	24	10	13	13	15	12	12	15	10	12	23	27
Co	4820	14	13	16	24	10	13	14	14	12	12	15	11	12	23	27
Év	2299	14	13	16	24	10	13	14	15	12	13	15	11	12	23	28
Év	2919	14	13	16	24	10	13	13	15	12	13	15	11	13	23	28
Év	2921	14	13	16	25	10	13	13	15	12	12	15	11	12	23	27
Év	2923	14	13	16	22	11	13	13	15	12	12	15	11	12	23	28
Év	2925	14	13	16	25	11	13	12	14	13	11	14	9	11	23	29
Év	2926	14	13	16	24	11	13	13	15	12	13	15	11	12	23	28
Év	2929	14	13	16	24	11	13	13	16	12	11	15	11	12	23	27
Év	2930	14	13	16	24	11	13	13	14	12	12	15	11	12	23	28
Év	2931	14	13	16	25	11	13	13	15	12	12	15	10	12	23	28
Év	2937	14	14	16	23	11	13	13	15	12	12	15	10	12	23	28
Fa	2599	14	15	16	24	11	13	13	15	12	12	15	10	12	23	28
Fa	2601	14	14	16	23	10	13	13	15	12	11	15	10	12	23	28
Fa	2602	15	13	16	24	10	13	13	15	12	12	15	10	12	23	27
Fa	2603	14	13	16	24	11	13	13	14	12	13	15	10	13	23	28
Fa	2605	14	12	16	24	11	13	13	15	10	13	15	9	12	23	27
Fa	2606	14	12	16	24	11	13	13	15	10	13	15	9	12	23	27
Fa	2608	14	13	16	24	10	13	13	15	12	12	15	11	12	23	28
Fa	2609	14	13	15	25	11	13	13	15	12	13	15	12	12	23	29
Fa	2611	13	13	16	24	11	13	13	16	12	12	15	11	12	23	27
Fa	2612	13	14	16	24	10	14	13	14	12	13	15	10	12	23	27
Fa	2613	14	13	15	24	10	13	13	15	12	11	14	11	12	23	28
Fa	2615	14	13	16	24	11	13	13	15	12	12	14	11	12	23	27
Fa	2616	13	14	16	23	11	13	13	15	12	11	15	11	12	24	28
Fa	2619	14	13	16	24	11	13	13	15	12	13	16	12	12	24	28
Gu	2451	14	13	16	25	12	13	12	15	13	11	14	10	11	23	29
Gu	2452	14	13	16	24	11	13	13	15	12	11	15	11	12	23	28
Gu	2454	13	14	16	23	10	13	14	15	9	9	14	11	10	21	27
Gu	2456	14	13	16	24	11	13	13	15	12	12	14	10	12	23	28
Gu	2460	14	15	16	24	11	13	13	15	12	12	15	11	12	23	28
Gu	2461	14	13	16	25	10	13	13	15	12	12	14	11	12	23	28
Gu	2462	14	13	16	23	11	13	13	15	12	12	14	10	12	23	27
Gu	2464	14	13	16	23	10	13	13	15	13	12	14	11	12	24	28

Gu	2467	14	13	17	24	11	13	13	15	12	14	14	11	12	23	28
Gu	2468	14	13	17	25	10	13	12	15	13	13	14	10	11	24	28
Gu	2473	14	13	16	24	11	13	13	15	12	12	15	10	12	23	27
Gu	2474	14	13	17	24	10	13	14	15	12	12	15	11	12	23	28
Gu	2475	14	13	16	24	10	13	13	15	12	12	15	11	12	23	27
Gu	2478	14	13	16	23	11	13	13	15	12	11	16	11	13	24	28
Gu	2479	14	13	14	23	10	13	13	15	12	12	16	11	12	23	28
Gu	2481	14	14	16	24	10	13	13	15	12	12	15	11	13	23	28
Le	811	14	13	16	24	10	13	12	14	12	13	15	11	12	23	28
Le	823	14	12	16	24	10	13	13	15	12	12	14	11	12	23	28
Le	1324	14	13	16	23	11	14	13	15	12	13	14	11	12	23	28
Le	2368	14	13	16	25	10	13	13	15	12	12	15	11	12	23	27
Le	2372	14	13	16	24	11	13	13	14	12	12	15	10	12	23	28
Le	2373	14	13	16	25	11	13	13	15	12	12	15	11	12	24	28
Le	2375	14	13	16	23	11	13	12	15	12	12	15	11	11	23	29
Le	2376	14	13	17	24	11	13	13	14	12	12	14	10	12	24	28
Le	2379	14	13	16	24	11	13	12	14	13	11	15	11	12	23	28
Le	2382	14	13	16	25	11	13	13	15	12	12	15	11	12	24	29
Le	2383	14	14	17	24	10	13	13	14	12	13	15	10	12	23	27
Le	2384	15	13	16	24	11	13	13	15	12	13	15	10	12	23	28
Le	2386	14	13	16	24	11	13	13	15	12	13	15	11	12	23	27
Le	2387	14	13	16	25	10	11	12	15	12	12	15	11	11	23	28
Le	2388	14	13	16	24	11	15	13	15	12	14	15	10	12	23	28
Le	2389	14	13	16	24	11	13	13	15	12	12	16	10	13	23	28
Le	2392	14	13	16	22	11	13	12	15	12	12	15	11	11	23	28
Le	3208	15	14	15	24	11	13	13	15	12	14	15	10	11	23	28
Li	1159	14	13	16	24	11	13	13	15	12	12	14	11	12	23	28
Li	1192	15	13	16	24	11	13	13	15	12	12	15	12	12	23	28
Li	2698	14	12	16	24	11	13	13	15	10	12	15	10	12	23	28
Li	2699	14	13	16	23	11	13	13	15	12	12	15	11	12	23	27
Li	2700	14	13	16	23	11	13	13	15	12	12	14	10	12	24	27
Li	2701	14	13	15	24	11	13	13	14	12	12	15	11	12	23	28
Li	2702	14	13	16	24	11	13	13	15	12	13	15	10	12	23	28
Li	2703	14	13	16	23	11	13	13	15	13	12	14	11	11	24	28
Li	2704	14	13	15	24	11	14	14	15	12	11	15	11	12	23	28

Li	2709	14	13	16	24	11	13	13	15	12	12	15	10	12	23	29
Li	2712	14	13	16	24	11	12	13	15	12	11	15	11	12	23	28
Li	2713	14	13	16	24	11	13	13	15	12	12	15	11	12	23	28
Li	2715	14	14	16	23	11	13	13	15	12	12	15	10	12	23	28
Li	2718	14	13	16	24	10	13	13	14	12	12	15	11	12	24	27
Li	2720	14	13	16	24	10	13	13	15	12	11	14	10	14	23	27
Li	2721	14	13	16	24	11	13	14	15	12	12	15	10	12	23	28
Li	2958	14	13	15	23	10	13	13	15	12	11	15	11	12	23	28
Li	3023	15	13	16	24	11	13	13	14	12	11	16	12	13	23	27
Li	3130	15	13	16	25	11	13	13	15	12	12	14	10	12	23	28
Po	2485	14	13	17	24	11	13	14	15	12	12	15	12	12	23	28
Po	2489	14	12	16	25	11	13	13	15	10	13	14	10	12	20	29
Po	2490	14	13	16	24	11	13	13	14	12	12	15	11	12	23	28
Po	2491	14	13	16	24	11	13	13	15	12	12	15	10	12	25	28
Po	2493	14	14	17	24	11	13	13	15	12	13	15	11	13	23	28
Po	2495	13	14	16	24	10	9	13	14	10	9	15	11	13	23	28
Po	2500	14	12	16	23	10	13	13	14	12	12	15	11	13	23	27
Po	2504	15	13	16	24	10	13	13	15	12	12	15	11	12	23	28
Po	2509	14	12	16	24	11	13	13	15	10	12	15	10	12	23	28
Po	2512	13	13	16	22	10	13	13	14	9	11	14	10	11	21	28
Sa	596	14	13	15	25	11	13	13	14	12	12	14	11	12	23	27
Sa	2434	15	13	14	24	11	13	13	15	12	11	15	10	12	24	28
Sa	4319	15	13	17	24	11	14	13	15	12	12	15	12	12	23	28
Se	156	14	13	16	24	11	13	13	15	12	11	15	11	12	25	28
Se	2290	13	13	16	24	10	13	13	15	12	11	15	11	12	23	28
Se	2291	14	13	16	23	11	13	13	15	12	12	14	11	12	23	27
Se	2292	14	15	16	24	11	13	13	15	12	12	15	11	12	23	28
Se	2294	13	13	16	24	11	13	13	15	12	12	14	11	12	23	28
Se	2295	14	13	16	25	11	13	13	14	13	15	15	11	12	23	27
Se	2296	15	14	16	24	11	13	13	15	12	11	15	10	12	23	28
Se	2298	14	13	16	23	11	13	13	15	12	12	15	11	12	23	27
Se	2301	14	13	16	25	10	13	13	15	12	11	15	10	11	23	28
Se	2302	15	13	16	24	11	13	13	15	11	12	15	11	13	23	28
Se	2304	14	13	14	23	10	13	13	15	12	13	16	11	12	23	28
Se	2305	14	12	16	24	11	13	13	15	12	12	14	10	12	23	28

Se	2306	14	13	15	25	11	13	13	15	12	11	15	11	12	23	28
Se	2309	14	13	16	24	11	13	13	15	10	13	15	10	12	24	28
Se	2310	14	13	16	25	11	13	13	14	12	12	15	10	12	24	28
Se	2312	14	13	16	24	10	13	14	15	12	12	14	11	12	23	28
Se	2313	14	13	16	24	10	13	13	15	12	11	15	11	12	23	27
Se	2315	15	14	16	24	11	13	13	15	12	11	15	11	12	23	28
Se	2317	15	14	16	24	11	13	13	16	12	12	15	10	12	23	28
Se	2318	14	14	16	24	11	13	13	15	12	12	15	10	12	23	28
Vi	241	14	13	16	24	10	13	13	15	12	15	15	11	12	24	27
Vi	489	15	13	16	24	11	13	13	15	10	12	15	10	12	23	28
Vi	919	14	14	16	24	10	13	13	15	12	12	14	11	12	23	28
Vi	1214	14	13	16	24	10	13	13	15	12	12	15	11	12	23	28
Vi	1251	14	14	16	23	11	13	13	15	12	12	15	11	12	23	28
Vi	1338	14	12	16	26	10	13	13	15	13	12	14	11	12	23	27
Vi	2020	13	13	16	24	10	13	13	15	12	11	14	11	13	23	26
Vi	2103	14	14	16	24	11	14	13	14	12	13	14	10	12	23	27
Vi	2115	14	14	16	24	11	13	13	14	12	12	14	11	12	24	28
Vi	2220	14	13	16	24	11	13	13	15	12	12	15	10	12	24	28
Vi	2330	14	13	18	24	10	13	13	15	12	12	15	10	12	24	29
Vi	2840	14	13	17	23	11	13	13	16	12	13	15	10	12	25	27
Vi	3380	14	12	16	24	11	14	13	15	12	11	15	10	12	24	28
Vi	3837	14	12	16	24	11	13	13	15	12	13	15	11	12	25	28
Vi	4191	14	13	16	24	11	13	13	15	12	13	16	11	12	23	28
Vi	4396	14	13	15	24	10	13	13	15	12	12	15	11	12	23	28
Vi	4839	14	13	16	24	11	13	13	15	12	11	15	11	12	26	28
Av	1292	14	13	16	24	11	13	13	15	11	12	15	11	13	24	28
Av	2801	14	13	15	24	11	13	13	15	12	12	15	11	12	23	28
CB	2644	14	13	16	23	10	13	13	15	12	12	15	11	12	23	28
Év	2805	14	13	17	23	11	13	13	15	12	12	15	11	13	23	28
Gu	2463	14	13	17	24	11	13	13	15	12	11	15	11	12	23	28
Le	2390	14	13	16	24	11	13	13	15	11	13	15	12	13	25	28
Li	2706	14	13	16	24	11	13	13	15	12	11	15	10	12	23	28
Li	2726	14	13	16	24	11	13	13	15	12	11	15	10	12	23	28
Vi	3959	15	13	16	24	11	13	13	15	12	12	15	11	12	23	28
Av	2580	14	13	17	24	11	13	12	15	12	13	15	11	12	23	27

Av	2844	14	13	16	23	11	13	13	15	12	13	15	11	12	23	28
Av	2849	15	13	18	24	11	13	13	14	12	12	14	11	12	23	28
Av	2852	14	13	16	26	10	13	13	15	12	12	15	11	12	23	27
CB	386	14	13	16	24	11	11	13	15	12	12	15	10	12	23	28
Év	2927	14	12	16	24	11	13	13	15	12	11	14	11	12	24	30
Fa	2617	14	13	16	24	12	13	13	15	12	11	14	12	12	23	27
Le	2380	16	13	19	25	11	11	13	14	11	10	16	12	11	23	28
Po	2483	15	13	16	24	11	13	12	15	12	14	15	10	12	23	27
Vi	2324	16	13	17	24	11	11	14	14	11	10	15	11	11	23	29
Av	498	14	13	17	24	11	13	13	15	12	13	15	12	12	23	28
Av	1543	14	13	15	25	10	13	13	15	12	12	15	10	12	24	28
Gu	2477	16	13	17	25	10	11	14	14	11	10	17	12	11	23	27
Li	2717	15	13	18	25	10	11	13	14	11	11	14	12	11	23	29
Li	2719	16	13	18	25	11	11	13	14	11	10	17	11	9	23	28
Vi	2355	15	13	17	25	10	11	13	14	11	11	14	12	11	23	29
Co	2797	14	14	16	23	11	14	12	16	11	12	16	10	11	22	27
Av	1240	13	13	16	23	10	13	14	15	9	12	14	11	11	20	28
Av	3167	15	14	17	23	10	13	12	14	9	11	15	10	11	22	27
Év	2936	15	14	16	23	11	13	13	14	9	11	14	10	11	22	27
Vi	1708	15	13	16	23	10	13	13	14	9	11	15	11	11	22	27
Gu	2459	15	13	16	23	10	13	14	14	9	11	14	11	11	21	27
Gu	2458	15	13	17	25	9	12	13	14	10	11	14	11	12	24	28
Av	1048	15	12	17	21	10	11	13	16	10	11	13	10	11	21	27
Av	1113	15	13	17	22	11	11	13	14	10	12	14	10	12	22	27
CB	2641	15	12	18	22	10	11	14	16	10	12	15	11	12	22	29
CB	2642	15	13	17	22	11	11	13	16	10	12	14	10	12	21	27
Co	2776	14	12	17	22	10	11	14	16	10	12	15	11	11	21	28
Év	2288	15	13	17	22	10	11	13	16	10	12	14	10	13	20	27
Év	2917	16	12	18	23	10	11	15	16	10	11	14	11	12	21	28
Év	2920	15	12	17	21	10	11	14	16	10	11	15	10	11	23	27
Év	3038	16	12	16	23	10	11	14	16	10	11	13	10	11	21	29
Fa	2604	16	13	18	24	10	11	14	16	10	11	13	9	11	21	27
Gu	2465	16	12	17	21	10	11	14	16	10	11	13	11	11	21	27
Gu	2470	15	12	18	22	10	11	14	16	11	11	14	10	11	20	27
Gu	2480	15	11	17	23	11	11	14	16	10	11	15	10	12	21	27

Le	2374	15	14	18	22	10	11	14	16	10	13	17	10	12	22	27
Le	2393	15	14	18	22	10	11	14	16	10	12	17	10	12	21	27
Le	2894	15	14	18	22	10	11	14	16	10	12	17	10	12	21	27
Li	2705	15	12	18	22	10	11	14	16	10	12	15	11	12	21	28
Li	2723	14	14	16	22	10	12	14	17	10	10	14	11	12	22	27
Po	2499	16	13	17	25	9	11	13	16	10	11	13	13	12	20	28
Se	2289	15	12	17	22	10	11	14	16	10	12	15	11	11	21	28
Av	1124	14	12	16	22	10	11	13	16	10	11	14	10	12	22	27
CB	2620	14	13	16	22	10	11	13	16	10	11	15	10	12	22	27
CB	3049	14	12	16	22	12	11	13	16	10	11	14	9	12	22	27
Co	1645	14	12	16	21	10	11	14	16	10	11	16	10	12	22	27
Év	2922	16	12	16	23	11	12	14	15	10	11	14	11	12	20	27
Év	2924	14	12	16	23	10	11	13	16	10	12	15	10	12	21	27
Év	2935	15	13	16	25	11	11	13	15	10	11	13	10	11	22	27
Fa	2600	14	12	16	22	10	11	13	16	10	11	15	11	12	22	27
Le	2367	15	13	17	24	10	12	15	15	10	11	15	11	12	19	27
Le	2391	14	12	18	22	10	11	13	16	10	13	14	11	12	21	27
Le	2394	15	13	17	24	10	12	15	15	10	11	15	11	12	19	27
Li	2716	14	12	17	23	10	11	13	16	10	12	15	10	12	22	27
Po	2484	14	12	16	23	10	11	13	16	10	11	15	10	12	21	27
Po	2486	14	12	16	23	10	11	13	16	10	11	15	10	12	21	27
Vi	4483	16	13	16	23	10	14	14	15	10	11	16	11	12	20	27
Av	1187	17	13	15	23	11	11	13	15	10	11	15	10	13	21	28
Be	3973	16	13	15	22	10	11	13	15	10	13	15	11	11	22	29
Év	2938	16	14	16	23	10	11	13	15	10	10	17	11	11	22	28
Le	2381	16	13	15	23	10	11	13	14	10	12	14	10	11	21	28
Sa	2510	17	13	15	23	11	11	13	15	10	11	15	11	14	21	28
Av	1077	14	14	17	23	10	11	12	14	10	12	15	11	10	20	27
Av	1483	14	13	16	25	10	11	12	14	10	11	14	11	11	21	27
Av	1941	14	13	18	25	10	11	13	14	10	11	15	11	11	21	27
Co	2774	14	13	18	22	10	11	12	14	10	12	15	11	11	20	27
Co	2793	14	13	18	23	10	11	12	14	10	12	15	11	10	20	27
Co	4636	14	13	18	24	10	11	12	14	10	11	15	11	11	20	27
Év	2928	17	12	17	24	10	11	13	16	10	11	14	11	13	21	26
Év	2932	13	12	17	24	10	11	12	14	10	12	14	10	11	21	26

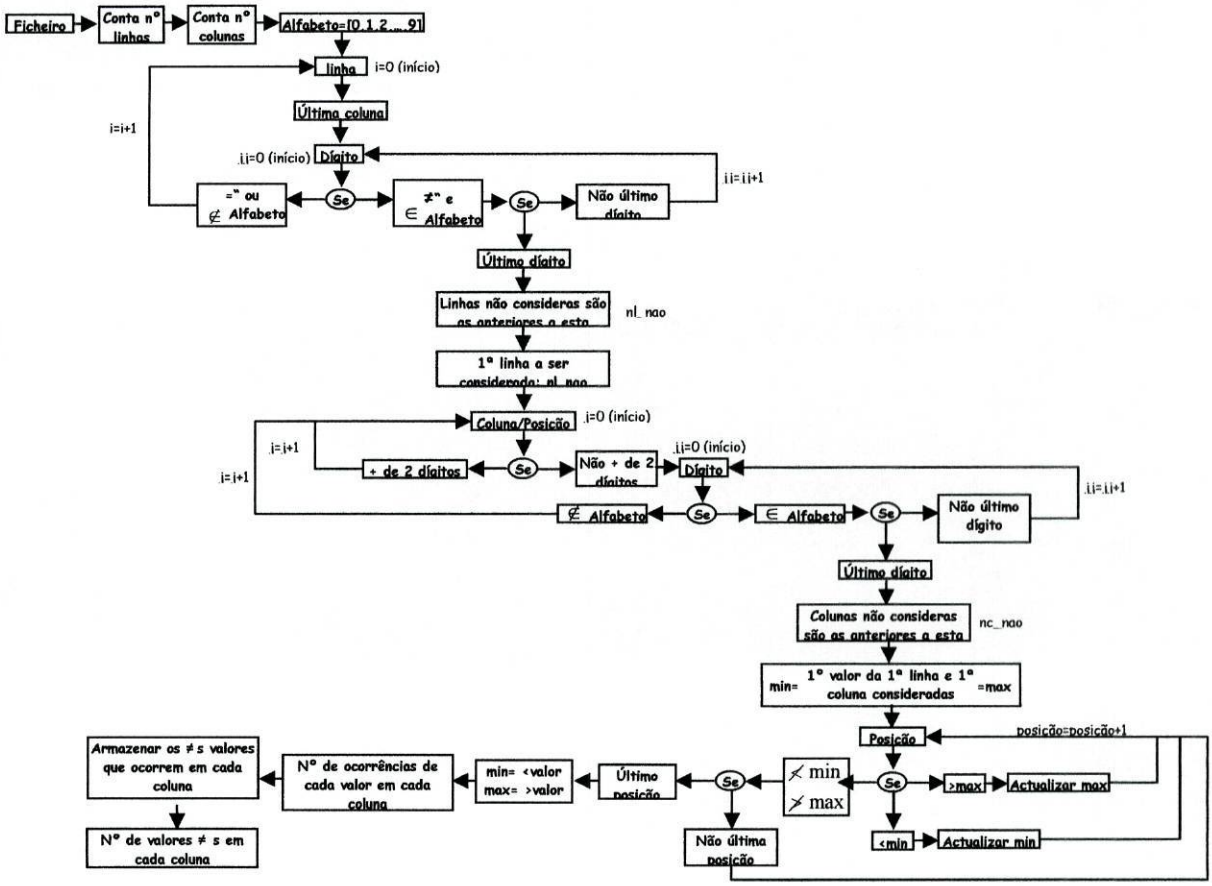
Év	2939	13	12	17	24	10	11	12	14	10	12	14	10	11	21	26
Fa	2618	14	13	17	23	10	11	12	14	10	11	14	10	12	21	27
Gu	2482	15	13	17	23	10	11	12	14	10	11	15	11	11	20	27
Po	2514	14	13	17	22	10	11	12	14	10	11	16	11	11	20	27
Se	2307	14	13	18	22	10	11	12	14	10	12	15	10	11	20	27
Av	1091	15	13	16	24	9	11	12	14	9	11	15	10	14	21	28
Av	1655	15	13	16	23	10	11	12	15	9	11	16	10	12	23	28
Be	590	14	13	17	23	10	11	12	15	9	11	15	10	12	21	29
Év	2934	13	13	16	23	10	12	12	15	9	11	14	10	13	21	28
Fa	2607	15	13	16	23	9	11	12	14	9	12	15	10	13	22	28
Fa	2610	15	12	17	25	10	11	12	17	9	11	14	11	10	22	27
Gu	2455	15	12	17	23	10	11	12	14	9	10	16	10	13	22	27
Gu	2476	14	13	16	24	10	11	12	15	9	12	15	10	12	21	28
Li	2707	14	13	16	23	10	11	13	15	9	12	15	11	13	24	27
Li	2708	15	12	16	24	10	11	13	17	9	11	14	11	10	21	27
Po	2494	15	13	19	22	10	11	12	14	9	11	15	9	12	23	27
Po	2497	14	13	16	23	10	11	12	14	10	10	15	10	12	22	27
Po	2498	14	13	16	23	10	11	12	14	10	10	15	10	12	22	27
Po	2515	14	12	17	24	11	11	12	16	9	11	14	11	10	21	27
Se	2314	15	12	16	24	10	11	12	16	9	12	15	10	10	21	27
Co	812	14	13	18	24	10	11	12	16	7	13	16	11	13	22	29
Co	4278	14	13	16	23	10	11	12	14	10	10	14	10	13	24	28
Li	2280	16	13	16	23	10	11	12	15	9	11	14	10	13	23	27
Sa	3971	14	14	17	23	10	11	13	14	9	11	14	9	13	23	27
Vi	2872	14	14	17	25	11	11	12	16	9	13	14	11	13	21	30
Vi	4021	14	14	17	25	11	11	12	16	9	13	14	11	13	21	30
Be	2297	16	11	17	21	10	11	13	14	11	13	14	11	13	22	28
Se	2316	17	12	17	22	9	12	13	16	10	12	14	10	13	20	28
Co	668	14	13	17	25	10	11	14	14	10	11	14	10	11	24	28
Le	2371	13	14	19	24	10	11	12	14	10	12	14	10	13	23	28
Li	1784	13	14	17	24	10	11	12	14	11	12	14	10	14	23	28
Se	2300	14	13	17	25	10	11	14	14	10	11	14	10	11	23	28
Av	1084	13	12	17	24	10	11	13	14	10	10	15	11	13	21	27
Av	1847	13	13	16	25	10	11	13	14	10	12	14	11	11	23	28
Av	3334	13	13	17	24	10	11	13	14	10	12	14	10	12	24	27

CB	2633	13	13	17	23	10	11	13	14	10	12	14	10	12	22	28
Év	2293	13	13	17	24	11	11	13	14	10	13	15	10	12	24	28
Év	2933	14	13	17	24	10	10	13	14	10	12	15	10	11	20	29
Gu	2453	13	13	17	24	10	11	13	14	10	12	16	10	12	21	28
Gu	2466	13	13	17	24	10	11	13	14	10	12	16	10	12	21	28
Gu	2469	13	13	18	23	10	11	13	14	10	12	14	9	12	23	28
Le	2378	13	13	17	24	10	11	13	14	10	11	15	10	12	24	28
Li	167	13	13	17	23	10	11	12	14	10	11	15	9	12	22	27
Po	2488	14	13	17	24	10	10	13	14	10	12	14	10	12	20	29
Po	2492	13	13	17	24	10	11	13	14	10	13	16	9	12	22	28
Po	2496	14	13	18	24	10	10	13	14	10	12	14	10	12	20	29
Po	2508	13	13	17	23	10	11	13	14	10	12	15	11	11	21	27
Sa	71	13	14	18	24	10	11	13	14	10	11	14	10	12	21	25
Sa	2319	13	13	18	25	10	11	13	14	10	11	14	10	11	22	28
Se	2303	15	13	18	24	10	11	13	14	10	11	14	12	12	20	28
Av	3293	13	14	16	24	9	11	13	14	10	10	14	11	13	21	8
Av	3296	13	14	16	24	9	11	13	14	10	10	12	13	14	21	28
Be	1195	14	14	17	23	9	11	13	14	10	10	14	10	13	21	28
CB	2621	13	14	17	24	9	11	14	14	9	11	14	11	12	22	28
CB	2625	13	14	16	24	9	11	13	14	10	10	14	11	13	21	28
Év	2918	13	13	16	24	9	11	13	14	10	11	14	10	13	21	28
Gu	2457	13	14	16	24	9	11	13	14	10	10	14	11	13	22	28
Le	2377	13	14	16	24	9	11	13	14	10	10	14	13	13	21	28
Le	2385	13	13	16	24	9	11	13	14	10	10	14	10	13	21	28
Li	536	13	14	16	24	9	11	13	14	10	10	14	11	13	21	27
Li	805	13	14	16	24	9	11	13	14	10	10	14	11	13	21	27
Li	2711	13	14	16	24	9	11	13	14	10	10	14	11	13	21	28
Li	2722	14	14	17	23	9	11	13	14	10	10	14	10	13	21	28
Po	2505	13	14	16	24	9	11	13	14	11	10	14	11	12	21	28
Po	2507	13	13	16	24	9	11	13	14	10	10	14	11	13	21	28
Po	2513	13	13	16	24	9	11	13	14	10	9	14	11	13	21	28
Vi	163	13	14	16	24	9	11	13	14	10	10	14	11	13	21	28
Vi	2274	13	14	16	24	9	11	13	14	10	10	14	11	13	21	29
Vi	3336	13	14	16	24	9	11	13	14	10	10	14	11	13	21	28
Vi	3666	13	14	16	24	9	11	13	14	10	10	14	11	12	21	28

Av	1968	13	13	18	25	10	11	13	14	10	12	14	10	13	22	27
Av	2984	15	13	18	23	10	11	14	14	11	11	15	10	11	23	27
Be	379	13	13	18	24	10	11	13	14	11	13	14	10	12	23	27
Co	1920	15	13	18	23	10	11	14	14	11	11	15	10	11	22	27
Fa	4266	14	14	16	24	11	13	13	15	12	12	15	11	12	23	28
Fa	4723	13	13	16	23	10	13	14	15	9	12	14	11	11	20	28
Fa	4823	16	13	15	22	13	11	13	15	10	13	14	11	11	23	28
Vi	4540	16	12	16	22	10	11	14	16	10	11	14	10	11	20	28
Vi	4655	14	13	16	23	11	13	13	15	12	12	15	11	13	23	28

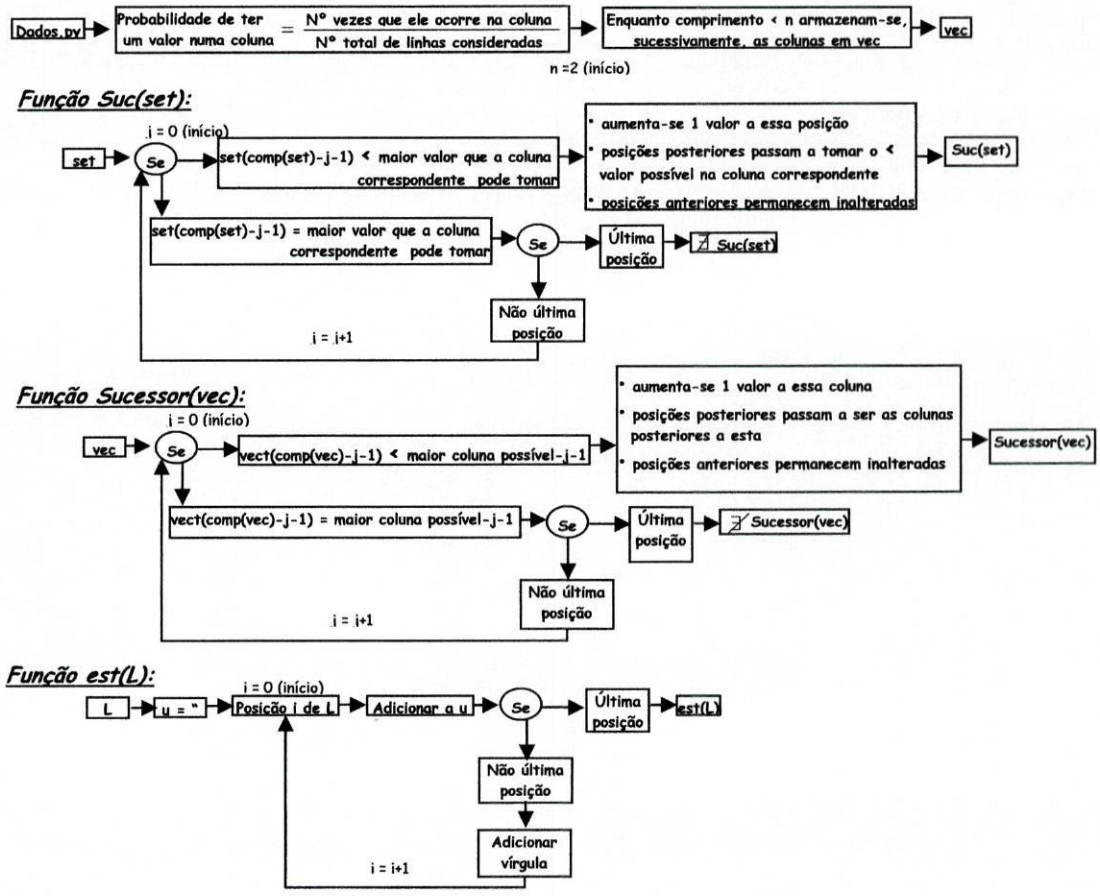
Apêndice C

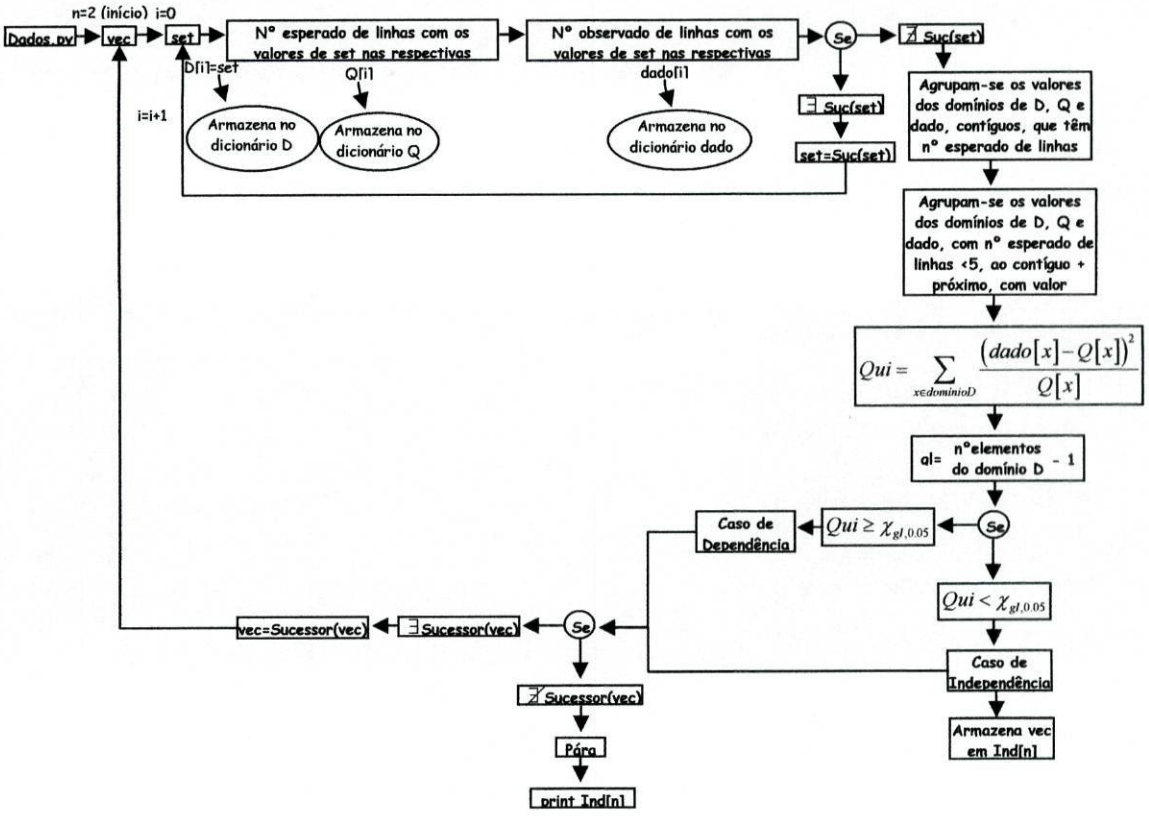
Esquema - Dados.py

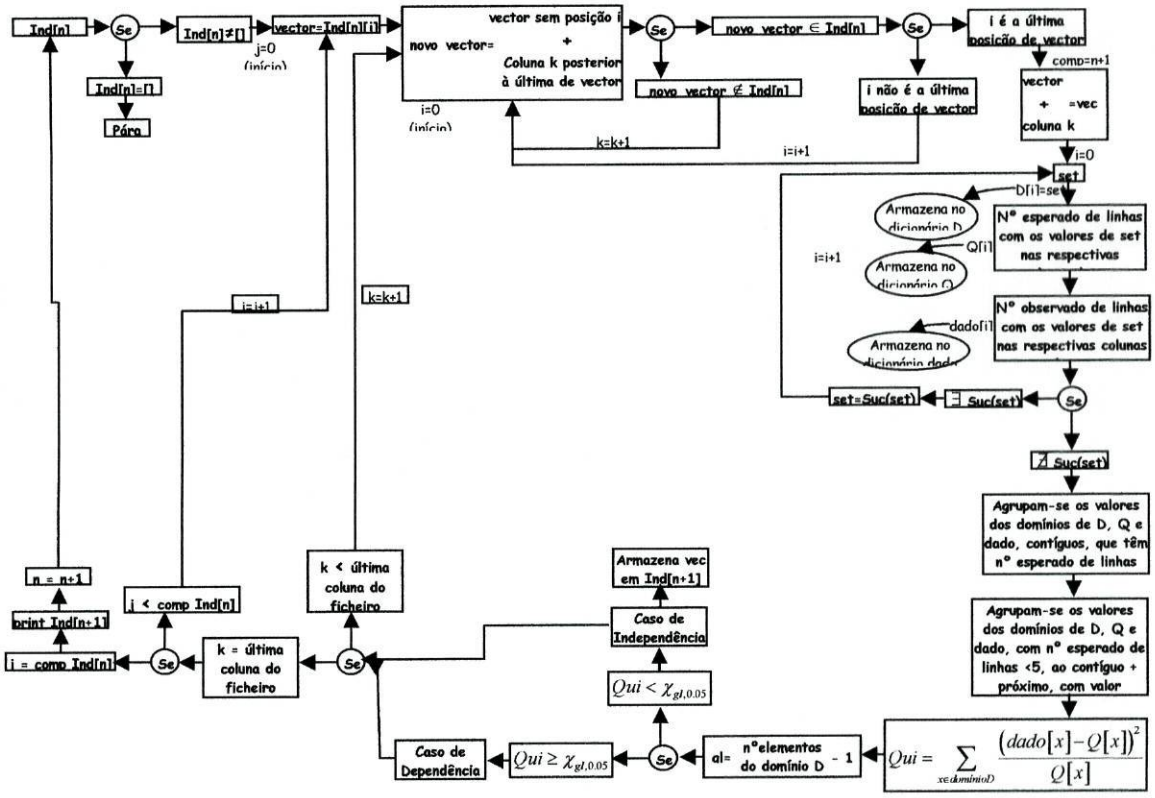


Apêndice D

Esquemas - Independência de 2 Colunas e Sucessivamente.py




n=2 (início) i=0
Dados.py
vec
set
Nº esperado de linhas com os valores de set nas respectivas
Nº observado de linhas com os valores de set nas respectivas
Se
∄ Suc(set)
D[i]=set
Q[i]
dado[i]
i=i+1
Armazena no dicionário D
Armazena no dicionário Q
Armazena no dicionário dado
∃ Suc(set)
set=Suc(set)
Agrupam-se os valores dos domínios de D, Q e dado, contíguos, que têm nº esperado de linhas
Agrupam-se os valores dos domínios de D, Q e dado, com nº esperado de linhas <5, ao contíguo + próximo, com valor
$Qui = \sum_{x \in domínioD} \frac{(dado[x] - Q[x])^2}{Q[x]}$
gl= nºelementos do domínio D - 1
$Qui \geq \chi_{gl,0.05}$
Se
Caso de Dependência
$Qui < \chi_{gl,0.05}$
Caso de Independência
Armazena vec em Ind[n]
vec=Sucessor(vec)
∃ Sucessor(vec)
Se
∄ Sucessor(vec)
Pára
print Ind[n]


Ind[n]
Se
Ind[n]≠[]
Ind[n]=[]
Pára
j=0
(início)
vector=Ind[n][j]
vector sem posição i
novo vector=
+
Coluna k posterior à última de vector
i=0
(início)
k=k+1
Se
novo vector ∈ Ind[n]
novo vector ∉ Ind[n]
Se
i é a última posição de vector
i não é a última posição de vector
comp=n+1
vector + =vec coluna k
i=i+1
i=0
set
D[i]=set
Armazena no dicionário D
Q[i]
Armazena no dicionário Q
i=i+1
N° esperado de linhas com os valores de set nas respectivas
N° observado de linhas com os valores de set nas respectivas colunas
dado[i]
Armazena no dicionário dado
∃ Suc(set)
Se
set=Suc(set)
∄ Suc(set)
Agrupam-se os valores dos domínios de D, Q e dado, contíguos, que têm n° esperado de linhas
Agrupam-se os valores dos domínios de D, Q e dado, com n° esperado de linhas <5, ao contíguo + próximo, com valor
$Qui = \sum_{x \in dominio D} \frac{(dado[x]-Q[x])^2}{Q[x]}$
gl= n°elementos do domínio D - 1
Se
$Qui \geq \chi_{gl,0.05}$
$Qui < \chi_{gl,0.05}$
Caso de Dependência
Caso de Independência
Armazena vec em Ind[n+1]
k < última coluna do ficheiro
Se
k = última coluna do ficheiro
Se
j < comp Ind[n]
i=i+1
k=k+1
j = comp Ind[n]
Se
print Ind[n+1]
n = n+1

Bibliografia

[1] Lutz, Mark (2001) *Programming Python* - 2nd Edition. O'Reilly. Beijing.

[2] Lutz, Mark; Ascher, David (1999) *Learning Python.* O'Reilly. Beijing.

[3] Bento, Murteira; Ribeiro, Carlos Silva; Silva, João Andrade e; Pimenta, Carlos (2002) *Introdução à Estatística.* McGraw-Hill. Lisboa.

[4] Wild, Christopher J.; Seber, George A. (2000) *Chance Encounters: A First Course in Data Analisis and Inference.* John Wiley and Sons. New York.

[5] Hines, William W.; Montgomery, Douglas C. (1980) *Probability and Statistics in Engineering and Management Science* - 2nd Edition. John Wiley and Sons. New York.

[6] Mood, Alexander M.; Graybill, Franklin A.; Boes, Duane C. (1913) *Introduction to the Theory of Statistics* - 3rd Edition. McGraw-Hill Series in Probability and Statistics.

[7] Pestana, Dinis Duarte; Velosa, Sílvio Filipe (2002) *Introdução à Probabilidade e à Estatística - Volume I.* Fundação Calouste Gulbenkian. Lisboa.

[8] Carreira, Adelaide; Pinto, Gonçalo; Sousa, Bruno (2002) *Cálculo da Probabilidade.* Instituito Piaget - Colecção Ciência e Técnica. Lisboa.

[9] Videira, Arnaldo (2001) *Engenharia Genética - Princípios e Aplicações.* Lidel - Edições Técnicas Lda. Lisboa, Porto, Coimbra.

[10] Stansfield, William D.; Colomé, Jaime S.; Cano, Raul J. (1998) *Biologia Molecular e Celular - Teoria e Exercícios.* Editora McGraw-Hill de Portugal, Lda. Amadora.

[11] Apontamentos da Disciplina *Genética Formal e Populacional* - Ano lectivo 2004/2005, fornecidos pelo Professor Doutor António Amorim.

[12] Morcilo, André Moreno (2004) *Teste do Qui-Quadrado.* Material de Apoio ao Curso de Pós-Graduação em *Saúde da Criança e do Adolescente* da *Faculdade de Ciências Médicas da UniCamp.* Campinas - SP - Brasil.