Universidade de São Paulo

Instituto Astronômico e Geofísico

Departamento de Astronomia

Sistemas em Absorção de QSOs e as Linhas de Estrutura Fina

Alex Ignácio da Silva

Orientador: Sueli M. M. Viegas

Tese apresentada à Universidade de São Paulo como requisito parcial à obtenção de grau de Doutor em Astronomia.

São Paulo, dezembro de 2003.

À minha querida esposa Goretti,

Agradecimentos

Chegar até aqui foi um caminho longo e árduo, porém extremamente recompensador. Cabe-me aqui registrar os meus agradecimentos àqueles que sempre estiveram comigo nessa jornada colaborando para o meu amadurecimento científico e pessoal:

Primeiramente à Sueli pelo estímulo e a dedicação de um trabalho de sete ou oito anos que resultou na minha formação como pesquisador. Com certeza se eu pude chegar tão longe foi porque a sua orientação me proporcionou condições para isso.

A todos aqueles que colaboraram com este trabalho fornecendo dados, em particular ao Manuel Bautista, pelas forças de colisão médias do Fe^{+} e seção de choque de fotoionização do Fe^{+3}, e também em especial à Sultana Nahar, a qual gentilmente nos cedeu para análise as seções de choque de fotoionização da matriz R calculados em seu grupo na Universidade Estadual de Ohio.

A todos os companheiros do instituto, professores e colegas alunos, pela longa convivência e tolerância com as minhas idiossincrasias. Certamente o ambiente no IAG é um dos mais amigáveis possíveis, o que fez a minha longa estadia aqui bastante prazerosa.

Aos meus pais, pelo suporte à minha educação e por terem moldado as influências que me levaram a um dia estudar física e me interessar por computação.

À Goretti, pela compreensão da ausência e pelo apoio em superar os revezes que surgiram pelo caminho.

A todos vocês o meu MUITO OBRIGADO!

Este trabalho tornou-se possível graças ao suporte financeiro da Fapesp, sob processo nº 99/05203-8.

The frozen sea surged around me,
It grew dark, the wind turned bitter, blowing
From the north, and the waves were savage. Creatures
Who sleep deep in the sea were stirred
Into life – and the iron hammered links
Of my mail shirt, these shining bits of metal
Woven across my breast, saved me
From death. A monster seized me, drew me
Swiftly toward the bottom, swimming with its claws
Tight in my flesh. But fate let me
Find its heart with my sword, hack myself
Free; I fought that beast's last battle,
Left it floating lifeless in the sea.
Other monsters crowded around me,
Continually attacking. I treated them politely,
Offering the edge of my razor-sharp sword.
But the feast, I think, did not please them, filled
Their evil bellies with no banquet-rich food,
Thrashing there at the bottom of the sea;
By morning they'd decided to sleep on the shore,
Lying on their backs, their blood spilled out
On the sand. Afterwards, sailors could cross
That sea-road and fell no fear; nothing
Would stop their passing. Then God's bright beacon
Appeared in the east, the water lay still,
And at last I could see the land, wind-swept
Cliff-walls at the edge of the coast. Fate saves
The living when they drive away death by themselves!
Lucky or not, nine was the number
Of sea-huge monsters I killed. (...)

Extraído do poema épico "*Beowulf*", anônimo, tradução de Burton Raffel, ed. Signet Classic, págs. 40-41.

Resumo

Neste trabalho estudamos os sistemas em absorção dos QSOs associados a galáxias interpostas. As condições físicas nestes sistemas são modeladas através de um código de fotoionização acoplado a um código de balanceamento estatístico de níveis atômicos.

Este último foi elaborado com uma estrutura bastante flexível de modo a permitir a sua utilização em diferentes aplicações, tais como no cálculo de razões de emissividades de linhas excitadas colisionalmente, ou ainda funções de resfriamento radiativo. Numa primeira fase, obtivemos as razões de povoamento dos níveis de estrutura fina do O^0 e Fe^+, complementando o trabalho desenvolvido no mestrado com os íons do C^0, C^+ e Si^+.

O código de fotoionização utilizado foi o Aangaba, para o qual foi feita uma revisão dos dados atômicos com o intuito de incorporar avanços recentes no conjunto de dados atômicos disponível na literatura. A fim de possibilitar o cálculo do equilíbrio de ionização de todos os elementos do H ao Zn, realizamos uma compilação de taxas de troca de carga (com hidrogênio e hélio) e ionização colisional. Também procuramos obter um conjunto auto-consistente de seções de choque de fotoionização e coeficientes de recombinação. Constatamos que tal conjunto de dados não está disponível na literatura, e, mesmo para os íons dos elementos mais abundantes, para os quais existem cálculos sofisticados da matriz R, os dados sofrem de uma grave deficiência relacionada à resolução das ressonâncias nas seções de choque.

Com o intuito de facilitar futuras atualizações, e permitir a adoção por outros autores em diferentes aplicações de interesse astrofísico, elaboramos uma estrutura simples e flexível para este novo banco de dados atômicos. Para projetar a sua implementação em *software*, lançamos mão de ferramentas modernas de programação: a programação orientada a objetos e a UML.

Como a maioria das aplicações em astronomia são escritas em Fortran, inclusive o Aangaba, esta foi então a linguagem escolhida para a implementação do banco de dados atômicos. Nesta fase pudemos identificar certas dificuldades relacionadas à implementação dos conceitos de programação orientada a objetos em Fortran. Também apontamos o fato de que o novo padrão da linguagem em processo de votação não incorporará o suporte direto à programação orientada a objetos de maneira plenamente satisfatória. Neste trabalho pudemos contornar estas dificuldades.

Finalmente, os dois códigos acima foram incorporados ao código de fotoionização Aangaba, e como uma primeira aplicação apresentamos uma grade de modelos para análise das razões de densidades de coluna dos níveis de estrutura fina derivadas nos sistemas Lyman-Limit. Essas razões de linhas podem fornecer pistas decisivas para desvendar a origem da fonte de ionização nestes sistemas.

Abstract

In this work we study QSO absorption systems associated with intervenient galaxies. The physical conditions in these systems are modelled through a photoionization code coupled to a statistical balance code of atomic levels.

The latter was designed with a very flexible structure in order to allow it to be used in different applications, such as the calculation of emissivity ratios of collisionally excited lines, or also radiative cooling functions. As a first step, we have obtained population ratios of the fine-structure levels of O^0 and Fe^+, complementing the work carried out earlier with the ions C^0, C^+ and Si^+.

The photoionization code employed was Aangaba, for which it was made a revision of the atomic data in order to incorporate recent advances in the atomic database available in the literature. In order to enable the calculation of the ionization equilibrium of all elements from H to Zn, we have made a compilation of charge exchange (with hydrogen and helium) and collisional ionization rates. We have also attempted to obtain a self-consistent dataset of photoionization cross sections and recombination rates. We have found that such a dataset is not available in the literature, and, even for the ions of the most abundant elements, for which there are sofisticated R-matrix calculations available, the data suffer from a serious shortcoming related to the resolution of the ressonances in the cross sections.

In order to facilitate future upgrades, and allow its widespread adoption by other authors in different astrophysical applications, we have designed a simple and flexible structure to this new atomic database. To design its software infrastructure, we have made use of modern programming tools: object-oriented programming and UML.

As most astronomical applications are written in Fortran, inclusive Aangaba, this was the language of choice to implement the atomic database. At this point we have spotted some difficulties related to the implementation of object-oriented concepts in Fortran. We also point out that the new language standard currently being voted will not incorporate direct support to object-oriented programming in a fully satisfactory way. In this work we were able to overcome these difficulties.

Finally, both codes above were incorporated to the photoionization code Aangaba, and as a first application we present a grid of models for the analysis of column density ratios of fine-structure levels derived in Lyman-Limit systems. These line ratios may provide fundamental clues to pin down the nature of the ionization source in these systems.

Índice

Agradecimentos **III**

Resumo **V**

Abstract **VI**

Índice **VII**

Figuras **X**

Tabelas **XIII**

Listagens **XIV**

1. Introdução **1**

1.1. Os sistemas em absorção de QSOs 1

1.2. Classificação dos sistemas em absorção 2

1.3. As linhas de estrutura fina 4

1.4. Plano geral da tese 5

2. As Equações de Equilíbrio Estatístico **9**

2.1. Introdução 9

2.2. As equações de balanceamento estatístico dos níveis 9

2.3. O código PopRatio 12

2.4. O equilíbrio estatístico do C^0, C^+ e Si^+ 14

2.5. O átomo O^0 16

2.6. O íon Fe^+ 19

3. O Banco de Dados Atômicos **28**

3.1. Introdução 28

3.2. Aspectos gerais 29

3.3. A sintaxe dos arquivos de dados atômicos 31

3.4. Benefícios e problemas 34

4. Troca de Carga e Ionização Colisional **39**

4.1. Troca de carga 39

4.2. Ionização colisional 45

5. Fotoionização/Recombinação **52**
5.1. Introdução 52
5.2. Coleta de dados 55
5.3. Suavização térmica 59
5.4. Coeficientes de recombinação unificados 62
5.5. Conclusões 72
6. Programação Orientada a Objetos **77**
6.1. Complexidade 77
6.2. Programação orientada a objetos e UML 79
6.2.1. Abstração 79
6.2.2. Encapsulamento 80
6.2.3. Herança 82
6.2.4. Polimorfismo 85
6.2.5. Outros conceitos 88
6.3. Padrões de projeto 91
7. Fortran Orientado a Objetos **95**
7.1. Introdução 95
7.2. Emulando POO em Fortran 90/95 96
7.2.1. Classes 98
7.2.2. Herança 103
7.2.3. Polimorfismo 106
7.2.4. Outras funcionalidades 109
7.3. Uma olhada no Fortran 2003 115
7.3.1. Classes 116
7.3.2. Herança e polimorfismo 118
7.3.3. Algumas omissões 120
7.4. Conclusões 122
8. A Implementação em Software do Banco de Dados Atômicos **126**
8.1. O projeto do banco de dados atômicos 126
8.2. Outras classes 131
8.3. A implementação 135
9. Os Sistemas Lyman-Limit **140**
9.1. Introdução 140
9.2. O sistema em $z_a = 2.652$ na direção de Q2231-00 142
9.3. A grade de modelos 143
9.3.1. O campo integrado dos QSOs 147
9.3.2. O campo do halo quente 148

9.4. Conclusão 159

10. Conclusões e Perspectivas 161

A. BDA: Ajustes e Exemplo 164

B. Algumas Funções Especiais 171

B1. $f_1(x) = e^x \int_1^\infty \frac{e^{-xt}}{t} dt$ 171

B2. $f_2(x) = e^x \int_1^\infty \frac{\ln(t)}{t} e^{-xt} dt$ 172

C. Códigos Fonte 176

C1. meuPacote_Ancestral_Class.f90 176

C2. meuPacote_Descendente_Class.f90 179

C3. meuPacote_Funcao_Interface.f90 182

C4. meuPacote_ClasseAbstrata_Class.f90 184

C5. meuPacote_ClasseConcreta_Class.f90 186

C6. meuPacote_Package.f90 188

C7. teste1.f90 189

C8. teste2.f90 190

C9. Ancestral.java 191

C10. Descendente.java 192

C11. Funcao.java 192

C12. ClasseAbstrata.java 193

C13. ClasseConcreta.java 193

C14. Teste1.java 194

C15. Teste2.java 195

Figuras

Figura 1.1: Espectro de um sistema em absorção Damped Lyman-α típico. 2

Figura 1.2: Espectro de um sistema em absorção Lyman Limit típico. 3

Figura 1.3: A floresta de Lyman-α. 3

Figura 1.4: Diagrama de níveis de energia do C^+ e as linhas de estrutura fina. 5

Figura 2.1: Taxas de excitação colisional para o C^0. 15

Figura 2.2: Taxas de excitação colisional para o C^+. 15

Figura 2.3: Taxas de excitação colisional para o Si^+. 16

Figura 2.4: Taxas de excitação colisional para o O^0. 18

Figura 2.5: Povoamento relativo dos níveis de estrutura fina $n(^3P_1)/ n(^3P_2)$ do O^0. 20

Figura 2.6: Povoamento relativo dos níveis de estrutura fina $n(^3P_0)/ n(^3P_2)$ do O^0. 21

Figura 2.7: Taxas de excitação colisional para o Fe^+. 22

Figura 2.8: Povoamento relativo dos níveis de estrutura fina do Fe^+: T = 1.000 – 5.000 K. 24

Figura 2.9: Povoamento relativo dos níveis de estrutura fina do Fe^+: T = 10.000 – 20.000 K. 25

Figura 4.1: Taxas de ionização por troca de carga com hidrogênio para o He^+. 41

Figura 4.2: Taxas de ionização por troca de carga com hélio para o Si^{+2}. 43

Figura 4.3: Taxas de recombinação por troca de carga com hélio para o Si^{+4}. 44

Figura 4.4: Taxas de ionização colisional para os íons do C. 49

Figura 5.1: Parâmetros dos ajustes dos coeficientes de recombinação radiativa dos íons do tipo Li e Be. 56

Figura 5.2: Seções de choque do OP suavizadas. 59

Figura 5.3: Coeficiente de recombinação unificado para o O III. 63

Figura 5.4: Seções de choque suavizadas para o Fe II. 64

Figura 5.5: Seções de choque suavizadas para o C II. 65

Figura 5.6: Seção de choque suavizada para o O I. 66

Figura 5.7: Seção de choque suavizada para o Ca VII. 66

Figura 5.8: Seções de choque suavizadas para o C IV e N V. 68

Figura 5.9: Seções de choque suavizadas para o O VI e Fe XXIV. 69
Figura 5.10: Seção de choque suavizada para o Al I. 70
Figura 5.11: Picos de ressonância mal-resolvidos na seção de choque do Fe II. 71
Figura 6.1: Modelagem do mundo real para UML para o software orientado a objetos. 82
Figura 6.2: Herança. 83
Figura 6.3: Associação entre classes. 84
Figura 6.4: Uma mensagem sendo enviada. 85
Figura 6.5: Hierarquia de classes em um aplicativo de desenhos. 85
Figura 6.6: Agregação por composição. 86
Figura 6.7: A relação de implementação. 88
Figura 6.8: Atributos e métodos estáticos. 89
Figura 6.9: Um pacote. 90
Figura 6.10: Resolvendo adequadamente a grade de regiões espaciais. 92
Figura 6.11: O padrão Composite. 93
Figura 6.12: Instâncias das células da grade de regiões espaciais. 93
Figura 7.1: Diagrama de classes ilustrando os conceitos básicos de POO. 98
Figura 7.2: Emulando os conceitos de POO em Fortran 90/95. 104
Figura 7.3: O padrão Template Method. 106
Figura 7.4: Uma associação bidirecional. 111
Figura 7.5: Um projeto que envolveria módulos recursivos em Fortran. 113
Figura 7.6: Implementação do padrão Composite em Fortran. 113
Figura 8.1: Classes básicas do BDA. 127
Figura 8.2: Improviso para implementar as classes básicas do BDA em Fortran. 127
Figura 8.3: Projeto completo do BDA. 128
Figura 8.4: Diagrama de estados da classe ADB. 131
Figura 8.5: Arquitetura em camadas de um código de fotoionização. 134
Figura 8.6: Telas exibidas após a execução da bateria de testes. 137
Figura 8.7: Refatoração sugerida para melhorar o projeto do BDA. 138
Figura 9.1: Campo de radiação do halo quente e integrado dos QSOs. 144
Figura 9.2: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do C^+ (QSOs de fundo). 149
Figura 9.3: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do Si^+ (QSOs de fundo). 150
Figura 9.4: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do Fe^+ (QSOs de fundo). 151

Figura 9.5: Tamanhos característicos e massas das regiões absorventes (QSOs de fundo). 152
Figura 9.6: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do C^+ (halo quente). 155
Figura 9.7: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do Si^+ (halo quente). 156
Figura 9.8: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do Fe^+ (halo quente). 157
Figura 9.9: Tamanhos característicos e massas das regiões absorventes (halo quente). 158

Tabelas

Tabela 2.1: Taxas de excitação dos níveis de estrutura fina do O^0 pela radiação cósmica de fundo. 17

Tabela 3.1: Possíveis formas de se organizar os dados no BDA. 33

Tabela 4.1: Atualizações às taxas de troca de carga com hidrogênio de Kingdon e Ferland (1996). 42

Tabela 4.2: Referências para reações de troca de carga com hélio publicadas na literatura desde Butler & Dalgarno (1980) e Butler et al. (1980). 43

Tabela 6.1: Exemplos de multiplicidade em UML. 84

Tabela 6.2: Especificadores de acesso. 91

Tabela 9.1: Densidades de coluna no sistema em $z_a = 2.652$ na direção de Q2231-00. 142

Tabela 9.2: Abundâncias dos sistema-solar. 145

Tabela B.1: Coeficientes p_n e q_n da expansão racional de $f_2(x)$. 174

Listagens

Listagem 6.1: A classe Ponto. 81
Listagem 6.2: A classe PontoColorido. 83
Listagem 6.3: Atributos e métodos estáticos. 89
Listagem 6.4: O padrão Singleton em Java. 92
Listagem 7.1: Uma “classe” em Fortran 95. 102
Listagem 7.2: O padrão Singleton em Fortran. 115

1. Introdução

1.1. Os sistemas em absorção de QSOs

Os chamados *objetos quase-estelares* foram descobertos no início da década de 60 como as contrapartidas no óptico de fontes intensas de rádio recém-descobertas, e esta denominação originou-se de suas aparências como fontes pontuais nas chapas fotográficas. A identificação de linhas fortemente deslocadas para o vermelho em seus espectros (Greenstein & Matthews 1963; Schmidt 1963; Greenstein & Schmidt 1964) levou ao estabelecimento da natureza extragaláctica desses objetos, bem como os qualificou como as fontes de energia mais intensas já descobertas. Por serem intrinsecamente brilhantes puderam ser detectados a enormes distâncias, tendo sido presentemente encontrados até *redshifts* $z \simeq 6$ (Fan et al. 2001).

Uma vez que os QSOs estão situados a distâncias cosmológicas, é bastante provável que a luz que emitem intercepte outros objetos situados ao longo da linha de visada durante a sua longa jornada até a detecção nos telescópios na Terra. Esses sistemas intervenientes absorverão parcialmente a luz do QSO de fundo em comprimentos de onda específicos, produzindo linhas em absorção no espectro.

A Figura 1.1 mostra um espectro típico: o contínuo relativamente plano e as linhas largas em emissão são produzidos pelo QSO em si, ao passo que as linhas estreitas em absorção são causadas por sistemas intervenientes, com a impressão digital característica dada pelo *redshift* em absorção menor do que o *redshift* em emissão.

Tais sistemas em absorção podem estar associados ao gás em galáxias de todos os tipos morfológicos, bem como ao gás difuso presente no meio intergaláctico. O estudo das linhas em absorção pode nos fornecer informações acerca da distribuição espacial, movimento, enriquecimento químico e estrutura de ionização destes sistemas em absorção. Uma vez que uma galáxia comum a *redshifts* tão elevados seria extremamente tênue, o QSO funciona como um "holofote de fundo", oferecendo a oportunidade de estudá-las indiretamente através das propriedades da absorção causada no espectro.

O estudo comparativo de vários sistemas em absorção em diferentes *redshifts* nos possibilita reconstruir a história do universo no intervalo $0 \leq z \leq 6$, desde quando se espera que tenham se formado as primeiras galáxias até os dias de hoje. É interessante notar que em $z = 6$ estamos observando o universo quando ele tinha menos de 10% de sua idade atual.

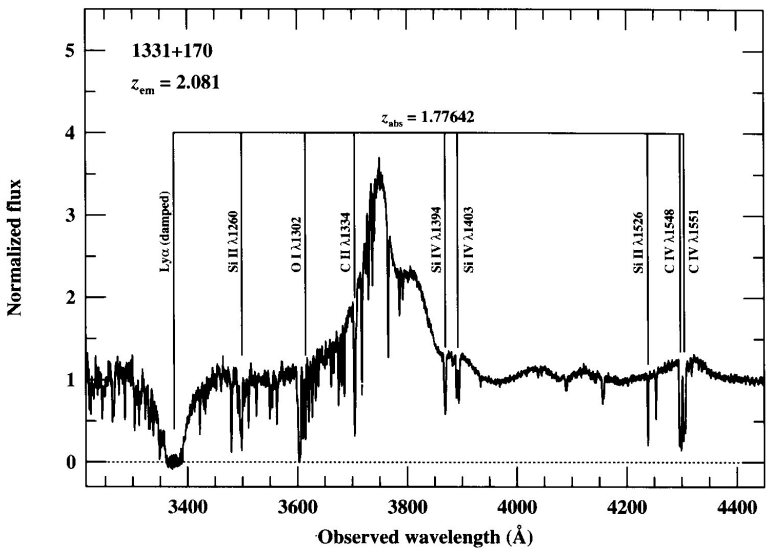


Figura 1.1: Espectro de um sistema em absorção Damped Ly-α típico.
Extraído de (Peterson 1997).

1.2. Classificação dos sistemas em absorção

Os sistemas em absorção dos QSOs costumam ser classificados de acordo com a densidade de coluna de hidrogênio neutro $N(\mathrm{H\ I})$ apresentada no espectro, a qual abrange pelo menos cerca de nove ordens de grandeza: $10^{12.5} \leq N(\mathrm{H\ I}) \leq 10^{22}\ \mathrm{cm}^{-2}$, sendo que o número de sistemas cai rapidamente com o aumento da densidade de coluna (Petitjean et al. 1993). Historicamente, convencionou-se distribuí-los em três categorias principais, conforme o valor de $N(\mathrm{H\ I})$ (Bechtold 1999; Charlton & Churchill 2001).

No extremo superior, sistemas com $N(\mathrm{H\ I}) > 10^{20.3}$ são denominados sistemas *Damped Lyman-α* (DLA), por possuírem a linha em absorção de Ly-α fortemente saturada. O sistema mostrado na Figura 1.1 pertence a esta categoria; nele podemos verificar que toda a radiação do QSO no centro da linha de Ly-α foi absorvida, restando apenas a absorção na asa com o perfil Lorenziano característico.

Sistemas com densidades de coluna intermediárias, $10^{17.2} \leq N(\mathrm{H\ I}) \leq 10^{20.3}$, são conhecidos como sistemas *Lyman Limit* (LL), e são caracterizados por uma quebra no espectro na região dos fótons que podem ionizar o H I, ou seja, aqueles com energia maior do que 13.6 eV. A Figura 1.2 mostra um espectro típico com a descontinuidade característica no limite da série de Lyman para o *redshift* do sistema em absorção.

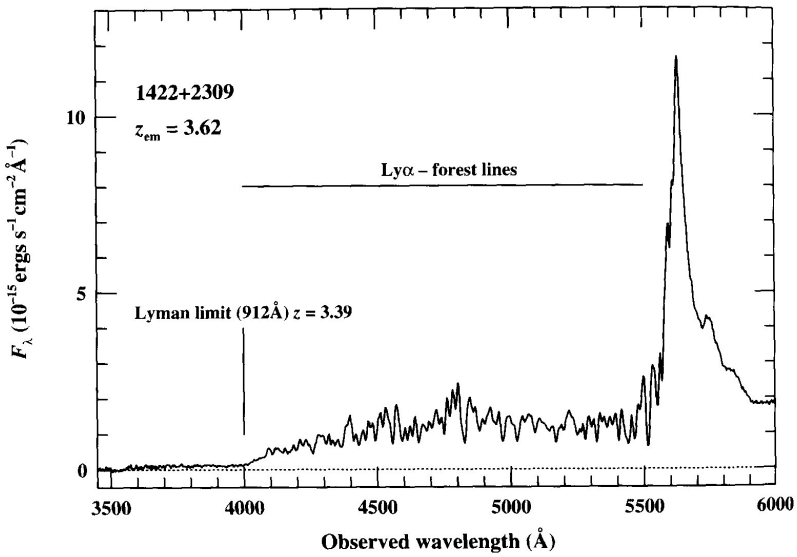


Figura 1.2: Espectro de um sistema em absorção Lyman Limit típico.
Extraído de (Peterson 1997).

Os sistemas com $N(\mathrm{H\,I}) < 10^{17.2}$ são de longe os mais numerosos, e pertencem à chamada *floresta de Ly-α*. No espectro da Figura 1.3 a linha em emissão à direita corresponde à transição de Ly-α do QSO, ao passo que as inúmeras outras linhas em absorção do lado azul desta são causadas pela mesma transição de Ly-α, porém cada qual situada a um *redshift* menor do que o *redshift* em emissão. Pelo fato de a linha de Ly-α ser uma transição ressonante, e devido à onipresença de hidrogênio no universo, teremos que esta região do espectro estará densamente povoada. Esta confusão de linhas dificulta enormemente a identificação de qualquer outra transição que porventura venha a cair nesta mesma região.

O estudo estatístico dos sistemas da floresta de Ly-α possibilita analisar o grau de formação de estruturas em grande escala do universo ao longo do espaço e do tempo (Rauch 1998).

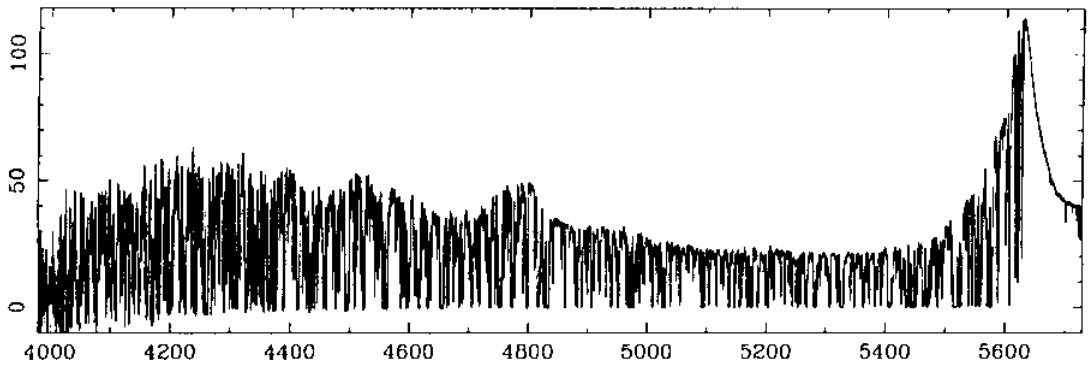


Figura 1.3: A floresta de Ly-α.
Extraído de (Rauch 1998).

1.3. As linhas de estrutura fina

Desde há muito tempo se sabe que os povoamentos dos níveis de estrutura fina de átomos/íons de elementos comuns podem servir como indicadores das condições físicas do meio onde se encontram (Bahcall & Wolf 1968; Smeding & Pottasch 1979).

Na maior parte dos ambientes astrofísicos, os átomos/íons se encontrarão quase que em sua totalidade povoados unicamente nos seus níveis de mais baixa energia. Entretanto se algum mecanismo de excitação estiver em ação, então uma fração pequena ou mesmo significativa destes átomos/íons serão encontrados também povoados nos níveis de energia imediatamente acima do fundamental. Se o mecanismo de excitação não for suficientemente intenso a ponto de manter os povoamentos dos níveis de energia em equilíbrio termodinâmico local, então teremos que o povoamento relativo dos mesmos poderá servir como indicador das condições físicas presentes.

Por exemplo, se colisões com elétrons são responsáveis pela excitação dos níveis de estrutura fina, então seus povoamentos relativos dependerão da densidade eletrônica do meio. Se, por outro lado, o mecanismo de excitação for devido à fótons no infra-vermelho, então o povoamento dos níveis poderá servir como indicador do fluxo do campo de radiação no infra-vermelho. Esse domínio de um mecanismo de excitação sobre outro é determinado não apenas pelas suas intensidades relativas, mas também por um jogo de vários parâmetros atômicos envolvidos, tais como probabilidades de transição, espaçamento dos níveis de energia, forças de colisão médias, etc.

É possível determinar o povoamento dos níveis de estrutura fina observando-se transições ultra-violeta em absorção no espectro dos QSOs, as quais serão deslocadas para a janela óptica com o *redshift* dos sistemas.

O diagrama de níveis de energia da Figura 1.4 ilustra a idéia para as linhas do C II. A maioria dos íons de C^+ se encontrará no estado fundamental $^2P_{1/2}$, e criará uma linha em absorção em λ = 1334.5 Å, cuja largura equivalente medida no espectro servirá para determinar a densidade de coluna de íons no estado fundamental N. Já os íons de C^+ que estiverem povoados no nível de estrutura fina excitado $^2P_{3/2}$ serão responsáveis por uma absorção bem próxima em λ = 1335.7 Å, cuja largura equivalente por sua vez possibilitará a determinação da densidade de coluna de íons no estado excitado N^*. Como a fração de íons povoados acima do fundamental será pequena, teremos que a linha proveniente do nível de estrutura fina excitado será bastante débil, o que na prática tem restringido a sua detecção apenas em observações feitas em espectrógrafos de alta resolução em telescópios de grande porte, que conseguem atingir uma relação sinal/ruído suficiente (Silva 1999).

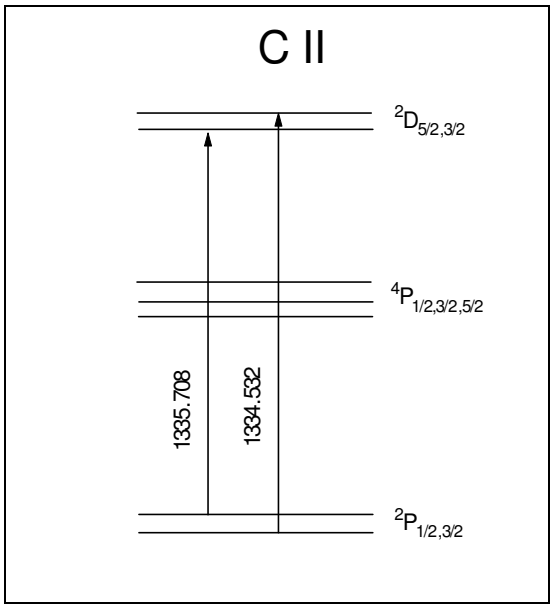


Figura 1.4: Diagrama de níveis de energia do C^+ e as linhas de estrutura fina.
Os níveis de estrutura fina não estão em escala. Extraído de Silva (1999).

Numa primeira aproximação, a razão entre as densidades de coluna dos níveis de estrutura fina será equivalente à razão entre as correspondentes densidades volumétricas:

$$\frac{N^*}{N} = \frac{n^*}{n} \quad . \tag{1.1}$$

O lado direito desta equação pode ser calculado teoricamente como função das condições físicas da nuvem resolvendo-se as equações de equilíbrio estatístico pertinentes. Assim, ao confrontar as razões das densidades de coluna observadas com as correspondentes razões das densidades volumétricas calculadas, podemos inferir quais são as condições físicas presentes na nuvem.

1.4. Plano geral da tese

Este trabalho é uma continuação direta daquele realizado em Silva (1999). Lá, resolvemos as equações de equilíbrio estatístico para calcular as razões de povoamento dos níveis de estrutura fina do C^0, C^+ e Si^+. Reunindo na literatura as razões de densidades de coluna correspondentes observadas nos sistemas em absorção dos QSOs, e confrontando-as com nossos cálculos teóricos através da equação (1.1), pudemos então inferir as condições físicas do meio nos sistemas DLA e LL.

Porém, para obter uma imagem mais precisa destes sistemas, é necessário empregar um tratamento mais sofisticado que consiste em modelá-los com um código de fotoionização apropriado a fim de obter as condições físicas camada a camada da nuvem. Neste caso a relação aproximada (1.1) seria substituída por:

$$\frac{N^*}{N} = \frac{\int n^*(r)dr}{\int n(r)dr} \quad . \tag{1.2}$$

Nosso objetivo final é então acoplar o código desenvolvido em Silva (1999) para o cálculo do balanceamento estatístico dos níveis de estrutura fina com um código maior de fotoionização, e em seguida aplicá-lo aos sistemas em absorção dos QSOs.

O capítulo 2 a seguir descreve a fase inicial de nosso trabalho, que consistiu em calcular as razões de povoamento dos níveis de estrutura fina do O^0 e Fe^+, a exemplo do que foi feito em Silva (1999) para o C^0, C^+ e Si^+. Também durante esta etapa criamos um novo código para calcular o equilíbrio estatístico de níveis de energia atômicos.

Antes de aplicar o código de fotoionização Aangaba aos sistemas em absorção dos QSOs, nos propusemos ainda a realizar uma atualização ao seu banco de dados atômicos. A atualização consistiu em compilar os dados atômicos necessários ao equilíbrio de ionização de todos os elementos do H ao Zn. O capítulo 4 descreve as fontes de onde foram extraídas as taxas de troca de carga (com hidrogênio e hélio) e ionização colisional, ao passo que as seções de choque de fotoionização e coeficientes de recombinação são tratados em pormenores no capítulo 5. Com o intuito de facilitar futuras atualizações como esta, idealizamos uma estrutura simples e flexível que permite que o nosso banco de dados atômicos seja facilmente estendido para incorporar novos dados que continuarão surgindo na literatura, a qual é apresentada no capítulo 3.

A fim de realizar com sucesso a implementação em *software* do nosso banco de dados atômicos, lançamos mão de ferramentas modernas de programação: a programação orientada a objetos e a UML. O leitor não familiarizado com este ferramental encontrará no capítulo 6 uma introdução sucinta a estes tópicos.

O projeto do banco de dados atômicos é descrito em detalhes no capítulo 8. Este foi implementado em Fortran 95, antecipando a sua fusão com o código de fotoionização de Aangaba, o qual é escrito em Fortran 77. A utilização de programação orientada a objetos em Fortran é uma questão à parte, a qual é tratada no capítulo 7 onde também apresentamos alguns resultados inéditos que obtivemos neste setor.

Por fim, nossos códigos de balanceamento estatístico e do banco de dados atômicos são incorporados ao código de fotoionização Aangaba para modelar os sistemas em absorção dos QSOs, com isto atingindo o objetivo final de nosso trabalho. A aplicação consistiu em elaborar uma grade de modelos para ser aplicada aos sistemas LL, a qual apresentamos no capítulo 9.

As conclusões gerais e perspectivas de trabalhos futuros seguem no capítulo 10.

Referências

Bahcall, J. N., & Wolf, R. A. 1968, ApJ, 152, 701

Bechtold, J. 1999, in *"Galaxies at high redshift"*, eds. I. Pérez-Fournon, M. Balcells, F. Moreno-Insertis, & F. Sánchez (Ilhas Canárias: Cambridge University Press), p. 131 (arXiv astro-ph/0112521)

Charlton, J. C., & Churchill, C. W. 2001, in *"Encyclopedia of Astronomy and Astrophysics"*, ed. P. Murdin, (Institute of Physics Publishing e Nature Publishing Group)(arXiv astro-ph/0006002) http://www.ency-astro.com

Fan, X., et al. 2001, AJ, 122, 2833

Greenstein, J. L., & Matthews, T. A. 1963, Nature, 197, 1041

Greenstein, J. L., & Schmidt, M. 1964, ApJ, 140, 1

Peterson, B. M. 1997, *"An Introduction to Active Galactic Nuclei"* (Cambridge: Cambridge University Press)

Petitjean, P., Webb, J. K., Rauch, M., Carswell, R. F., & Lanzetta, K. 1993, MNRAS, 262, 499

Rauch, M. 1998, ARA&A, 36, 267

Schmidt, M. 1963, Nature, 197, 1040

Silva, A. I. 1999, dissertação de mestrado, *"Linhas de estrutura fina em absorção no espectro de QSOs."* (Instituto Astronômico e Geofísico, Universidade de São Paulo, São Paulo) http://www.ignacioalex.com/bibliografia

Smeding, A. G., & Pottasch, S. R. 1979, A&AS, 35, 257

2. As Equações de Equilíbrio Estatístico

2.1. Introdução

O objetivo deste capítulo é mostrar como é feito o cálculo teórico do povoamento dos níveis de estrutura fina como função das condições físicas do meio.

Na seção 2.2 expomos as equações básicas que governam o equilíbrio estatístico dos níveis, e em seguida na seção 2.3 falamos sobre o código desenvolvido para resolvê-las numericamente. Os átomos/íons considerados foram o C^0, C^+, O^0, Si^+ e Fe^+. Estes cinco átomos/íons foram selecionados previamente (Silva 1999) por serem os únicos a apresentarem o termo fundamental subdividido em níveis de estrutura fina, e por apresentarem transições ressonantes com comprimento de onda superior à transição de Ly-α do hidrogênio em 1216 Å (de tal forma que elas nem sempre cairão na região da floresta de Ly-α do espectro). Os dois primeiros íons já tiveram as linhas de estrutura fina observadas em sistemas em absorção intervenientes (Silva 1999; Silva & Viegas 2002), ao passo que os dois últimos foram detectados em alguns sistemas em absorção associados ao QSO (Wampler, Chugai, & Petitjean 1995; Srianand & Petitjean 2000; Hamann et al. 2001; Kool et al. 2001; Srianand & Petitjean 2001).

Na seção 2.4 discorremos brevemente acerca do C^0, C^+ e Si^+. Uma vez que estes átomos/íons já foram considerados em Silva (1999), nos limitamos aqui a apenas relembrar alguns resultados principais que serão necessários posteriormente. Por fim, nas seções 2.5 e 2.6 apresentamos em detalhes os novos resultados obtidos para o O^0 e Fe^+.

O código desenvolvido para resolver as equações de equilíbrio estatístico, juntamente com os arquivos de entrada correspondentes aos modelos atômicos para os átomos/íons considerados, se encontra disponível no CD em anexo.

2.2. As equações de balanceamento estatístico dos níveis

A fim de calcular o povoamento dos níveis de um dado átomo/íon, assumimos como válidas duas premissas básicas:

1. Processos que envolvem outros estágios de ionização além do átomo/íon considerado (tais como fotoionização, recombinação, reações de troca de carga, ionização colisional, etc.) são lentos se comparados aos processos ligado-ligado,

2. Todas as transições consideradas são opticamente finas.

Para obter o povoamento de um dado nível *i*, devemos levar em consideração todos os possíveis processos que o povoam e o despovoam:

$$\frac{dn_i}{dt} = \sum (\text{processos que povoam}) - \sum (\text{processos que despovoam}) \quad , \tag{2.1}$$

onde n_i é a densidade volumétrica de átomos/íons no nível *i*.

Portanto, no estado estacionário teremos que a soma de todos os processos que povoam um dado nível será equilibrada pela soma de todos os processos que o despovoam. Supondo que as duas condições acima são verdadeiras, isso pode ser expresso da seguinte forma:

$$\sum_j n_j Q_{ji} = n_i \sum_j Q_{ij} \quad . \tag{2.2}$$

O lado esquerdo é uma soma sobre todos os processos que povoam o nível *i* a partir dos outros níveis *j*, enquanto que o lado direito é uma soma sobre todos os processos que despovoam o nível *i* para os níveis *j*. Os coeficientes Q_{ij} serão dados por (veja, por exemplo, Rybicki & Lightman 1979):

$$Q_{ij} = A_{ij} + B_{ij} u_{ij} + \Gamma_{ij} + \sum_k n^k q_{ij}^k \quad , \tag{2.3}$$

onde foram considerados todos os processos ligado-ligado possíveis, *i.e.*, espontâneos, induzidos por radiação ou induzidos por colisões. Por definição fizemos $A_{ii} = B_{ii} = u_{ii} = \Gamma_{ii} = q_{ii} = 0$.

A_{ij} é a probabilidade de transição do decaimento espontâneo do nível *i* para o nível *j*. Para $i \leq j$, $A_{ij} = 0$.

B_{ij} são coeficientes de Einstein, relacionados às probabilidades de transição, para $i > j$, por:

$$\begin{aligned} B_{ij} &= \frac{1}{8\pi h} \frac{A_{ij}}{(E_i - E_j)^3} \\ B_{ji} &= \frac{g_i}{g_j} B_{ij} \quad , \end{aligned} \tag{2.4}$$

onde *h* é a constante de Planck, E_i é a energia do nível *i* (expressa em cm^{-1}) e g_i é o peso estatístico do nível *i*.

u_{ij} é a densidade de energia espectral do campo de radiação integrada ao longo do perfil da linha ϕ_ν, correspondente à transição do nível *i* para o nível *j*:

$$u_{ij} = u_{ji} = \int u_\nu \phi_\nu d\nu \simeq \int u_\nu \delta(\nu - \nu_{ij}) d\nu = u_\nu(\nu_{ij}) \quad . \tag{2.5}$$

ν_{ij} é a freqüência da transição e assumimos que o campo de radiação não varia significativamente ao longo do perfil da linha.

Γ_{ij} são as taxas de excitação indireta por fluorescência (Silva 1999; Silva & Viegas 2001):

$$\Gamma_{ij} = \sum_\lambda B_{i\lambda} u_{i\lambda} \frac{A_{\lambda j} + B_{\lambda j} u_{\lambda j}}{\sum_g (A_{\lambda g} + B_{\lambda g} u_{\lambda g})} \quad , \tag{2.6}$$

ou seja, temos a situação em que o átomo/íon inicialmente povoado no nível *i* pertencente aos *g* níveis de energia mais baixos, absorve um fóton ultravioleta e é levado a um nível superior λ, posteriormente decaindo (tanto espontaneamente quanto por emissão estimulada) de volta a um nível *j* também pertencente aos *g* níveis mais baixos. Como este processo pode se dar por intermédio de qualquer um dos níveis superiores λ, teremos que a taxa total de excitação indireta será uma soma sobre estes níveis.

Na equação (2.3) também levamos em conta o efeito de colisões com partículas do meio; n^k é a densidade volumétrica da partícula que induz a colisão, sendo que os parceiros de colisão mais importantes em geral são k = e^-, p^+, H^0, He^0, H_2, ... , dependendo se o meio está essencialmente ionizado ou neutro.

q_{ij}^k é a taxa de excitação colisional do nível *i* para o nível *j* induzida pelo parceiro de colisão *k*. Estes coeficientes nada mais são do que as seções de choque do processo relacionado σ_{ij} convoluídas com uma distribuição Maxwelliana de velocidades $f(v)$, tornando estas quantidades apropriadas para aplicações astrofísicas (veja, por exemplo, Osterbrock 1989):

$$q_{ij} = \langle v\sigma \rangle = \int v\sigma(v) f(v) dv = \frac{1}{(kT)^2} \sqrt{\frac{8kT}{\pi\mu}} \int_0^\infty \sigma_{ij}(\varepsilon) \varepsilon e^{-\varepsilon/kT} d\varepsilon \quad , \tag{2.7}$$

para as taxas de desexcitação ($i > j$); *k* é a constante de Boltzmann, *T* é a temperatura cinética, μ é a massa reduzida do sistema e ε é a energia cinética do parceiro de colisão.

Taxas de excitação e desexcitação estão relacionadas pelo princípio do balanceamento detalhado:

$$q_{ji} = \frac{g_i}{g_j} e^{-(E_i - E_j)/kT} q_{ij} \quad . \tag{2.8}$$

Quando a interação é Coulombiana, como é o caso de colisões com elétrons, é conveniente expressar a seção de choque em termos da *força de colisão* Ω_{ij} , definida por:

$$\sigma_{ij}(\varepsilon) = \frac{h^2}{8\pi m\varepsilon} \frac{\Omega(\varepsilon)}{g_i} \quad , \tag{2.9}$$

onde m é a massa do elétron. Substituindo esta na equação (2.7) obtemos:

$$q_{ij}^{e^-} = \frac{h^2}{mg_i} \sqrt{\frac{1}{8\pi^3 \mu kT}} \int_0^\infty \Omega_{ij}(\varepsilon) e^{-\varepsilon/kT} d\left(\frac{\varepsilon}{kT}\right) = \frac{8.629\times10^{-6}}{\sqrt{T}} \frac{\gamma_{ij}}{g_i} \quad , \tag{2.10}$$

com T expresso em K e γ_{ij} é definida pela integral na equação (2.10) e denomina-se *força de colisão média*[⁋]. Tipicamente γ_{ij} é uma função que varia lentamente com T, da ordem da unidade.

Estes são os parâmetros básicos necessários para se calcular o balanceamento estatístico dos níveis de um dado átomo/íon. Se em nosso modelo levarmos em consideração n níveis de energia, o sistema de equações (2.2) nos fornecerá $n-1$ equações independentes, as quais poderão ser resolvidas para o povoamento relativo dos níveis.

2.3. O código PopRatio

Com o intuito de calcular o balanceamento estatístico dos níveis, elaboramos um código em Fortran que resolve numericamente o sistema de equações (2.2). Tal procedimento é necessário, pois freqüentemente em nossas aplicações inúmeros processos de excitação contribuem para o povoamento dos níveis. Além disto, alguns átomos/íons possuem uma estrutura eletrônica complexa o suficiente para requerer a inclusão de vários níveis em sua descrição, tais como o ferro.

Embora anteriormente já dispuséssemos de um código similar, optamos por reescrevê-lo integralmente, pois com a prática e a experiência passamos a adotar um estilo mais claro de programação. Procuramos buscar inspiração na *linguagem F*, que nada mais é do que um

---

[⁋] Em inglês, *Maxwellian-averaged collision strength*.

subconjunto do Fortran 90 que exclui todas as construções arcaicas legadas do Fortran 77 e que foram mantidas no novo padrão da linguagem tão somente para proteger o investimento em códigos já existentes.

O novo código - batizado de PopRatio (Silva & Viegas 2001) - é bastante flexível, permitindo a inclusão de um número arbitrário de níveis de energia e mecanismos de excitação. Toda a memória necessária é alocada dinamicamente em tempo de execução.

Foram empregadas inúmeras rotinas das bibliotecas BLAS (Dongarra et al. 1988; Dongarra et al. 1990), LAPACK (Anderson et al. 1999), LAPACK90 (Blackford et al. 1996) e PPPACK (De Boor 1978), as quais podem ser obtidas gratuitamente na WWW (Netlib). Para conveniência do usuário, todas as rotinas necessárias são redistribuídas juntamente com o código.

Os parâmetros atômicos são lidos de um arquivo texto separado, cuja sintaxe (descrita em detalhes em Silva & Viegas 2001) permite a inclusão de comentários documentando a fonte de onde foram extraídos os dados.

O campo de radiação pode ser estipulado para uma forma arbitrária qualquer definida pelo usuário. Também é possível incluir alguns campos de radiação pré-definidos, tal como o campo proveniente da radiação cósmica de fundo, caracterizado por um espectro de corpo negro:

$$u_{ij} = \rho(\nu_{ij}, T) = \frac{8\pi h}{c^3} \frac{\nu_{ij}^3}{e^{h\nu_{ij}/kT} - 1} \quad , \tag{2.11}$$

onde c é a velocidade da luz. A temperatura dependerá do *redshift* de acordo com a relação prevista pelo modelo padrão do *Big Bang* (veja, por exemplo, Kolb & Turner 1990):

$$T = T_0(1+z) \quad , \tag{2.12}$$

onde $T_0 = 2.728 \pm 0.002$ K é o seu valor atual (Fixsen et al. 1996; Smoot & Scott 1998).

Em virtude da flexibilidade adicional proporcionada pelo novo código, ele agora pode ser estendido a outros problemas além das linhas de estrutura fina em absorção, podendo ser também aplicado no cálculo de emissividades de linhas excitadas colisionalmente e também no cálculo de funções de resfriamento. Os diversos usos do código são ilustrados em cinco programas de exemplo, cujas saídas podem ser utilizadas pelo usuário para validá-lo em sua plataforma.

2.4. O equilíbrio estatístico do C^0, C^+ e Si^+

Por possuírem os níveis de estrutura fina relativamente próximos uns dos outros em comparação com outros átomos/íons, o C^0 e o C^+ estão entre as espécies mais propensas a terem as linhas em absorção provenientes dos níveis excitados observadas nos espectros dos QSOs (Silva 1999; Silva & Viegas 2002).

A detecção do C^0 porém, está longe ser tarefa fácil e rotineira, já que este possui um potencial de ionização inferior ao do hidrogênio em 13.6 eV, e portanto terá uma abundância fracional baixíssima no gás. A observação das linhas de estrutura fina do C I demanda espectros com longos tempos de exposição em telescópios de grande porte a fim de atingir a relação sinal/ruído necessária.

Esforços consideráveis têm sido investidos no sentido de contornar esta dificuldade, pois o povoamento dos níveis de estrutura fina do C^0 é significativamente afetado pela radiação cósmica de fundo. Dos cinco átomos/íons apontados anteriormente, o C^0 é o que possui o menor espaçamento entre os níveis de estrutura fina, tornando-o o "termômetro cósmico" ideal para investigar a validade da lei de temperatura prevista pelo modelo padrão, equação (2.12), e confrontá-lo com modelos alternativos que prevêem um relação mais geral (Lima, Silva, & Viegas 2000):

$$T = T_0(1+z)^{1-\beta} \qquad 0 \le \beta \le 1 \ . \tag{2.13}$$

A Figura 2.1 mostra as taxas de excitação colisional para o primeiro nível de estrutura fina excitado do C^0 para os parceiros de colisão mais importantes.

Os íons C^+ e Si^+ apresentam configurações eletrônicas similares, o que se traduz num comportamento das curvas das taxas de colisão como função da temperatura bastante semelhantes, como podemos ver nas figuras 2.2 e 2.3[▷]. Em geral elétrons serão os parceiros de colisão mais importantes para estes íons, com átomos de hidrogênio neutro competindo apenas se o gás estiver bastante recombinado $n_{\mathrm{H}^0} > 100\, n_{\mathrm{e}^-}$; prótons só passam a agir em temperaturas altíssimas $T \approx 10^5$ K.

Para maiores detalhes acerca do povoamento dos níveis de estrutura fina do C^0, C^+ e Si^+ o leitor é remetido a Silva (1999) e Silva e Viegas (2002).

[▷] Note que em Silva (1999) não havíamos incluído colisões com prótons para o Si^+.

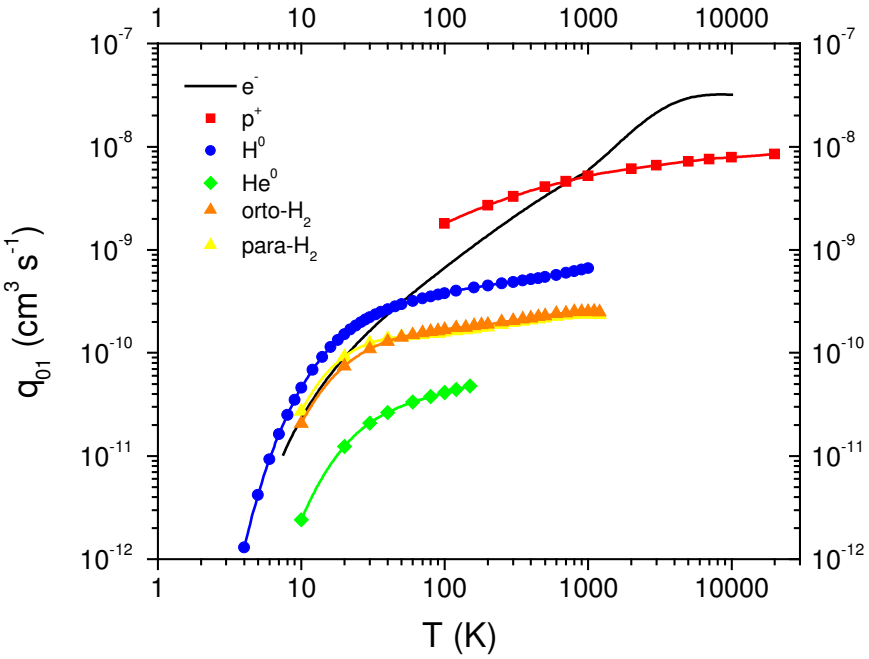


Figura 2.1: Taxas de excitação colisional para o C^0.
Taxas de excitação colisional entre os dois primeiros níveis de estrutura fina $^3P_0 \rightarrow {}^3P_1$ do C^0 para vários parceiros de colisão. Adaptado de Silva (1999).

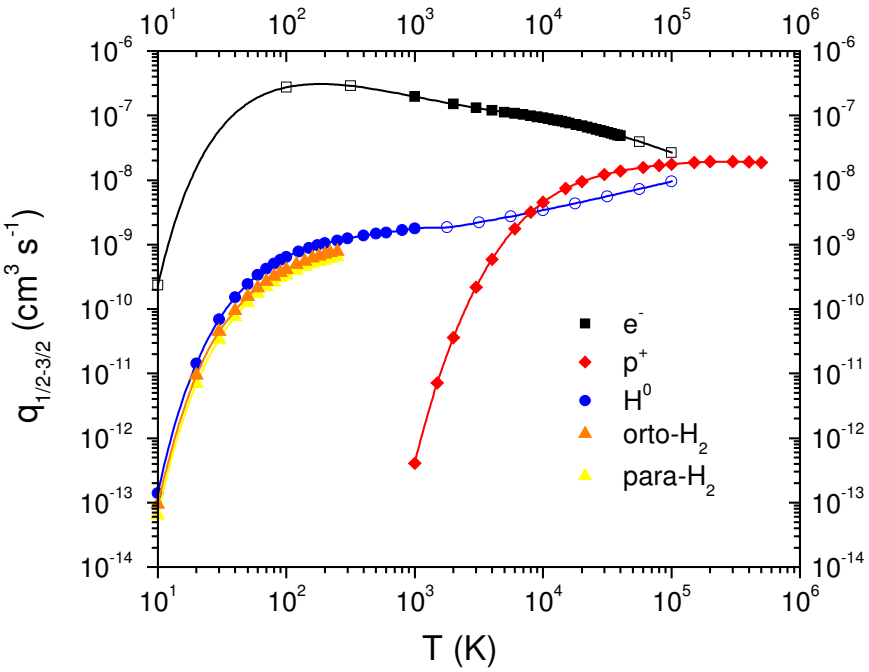


Figura 2.2: Taxas de excitação colisional para o C^+.
Taxas de excitação colisional entre os níveis de estrutura fina $^2P_{1/2} \rightarrow {}^2P_{3/2}$ do C^+ para vários parceiros de colisão. Adaptado de Silva (1999).

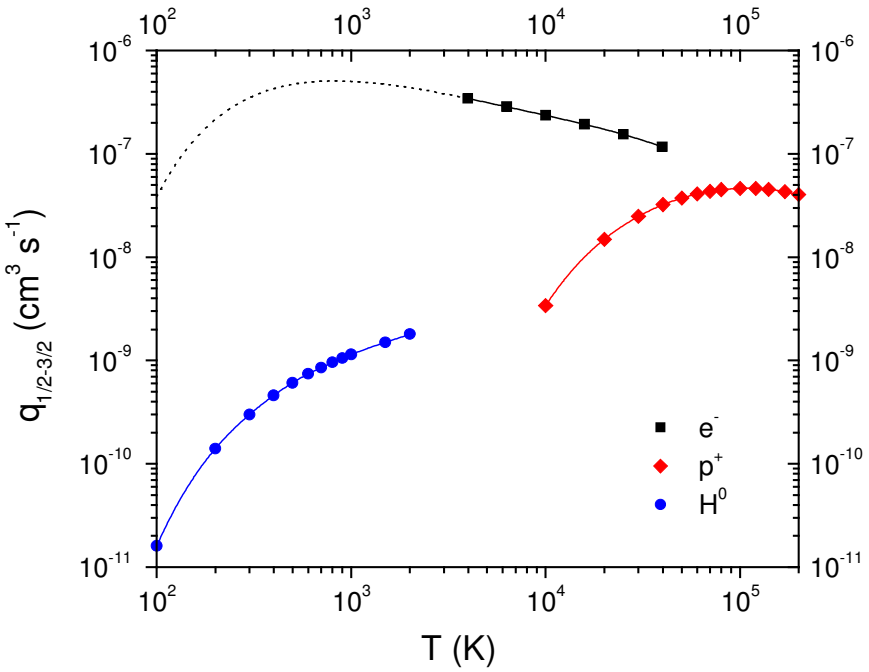


Figura 2.3: Taxas de excitação colisional para o Si^+.
Taxas de excitação colisional entre os níveis de estrutura fina $^2P_{1/2} \rightarrow {}^2P_{3/2}$ do Si^+ para vários parceiros de colisão. A curva pontilhada denota uma extrapolação da taxa de excitação colisional eletrônica assumindo a força de colisão média como constante. Adaptado e estendido de Silva (1999).

2.5. O átomo O^0

O nível fundamental do átomo de O^0 é composto pelo tripleto $2s^2\ 2p^4\ {}^3P_{2,1,0}$. As energias dos níveis de estrutura fina excitados em relação ao estado fundamental são 158,265 e 226,977 cm^{-1}. As probabilidades de transição são $A_{12} = 8,865\times10^{-5}\ s^{-1}$, $A_{02} = 1,275\times10^{-10}\ cm^{-1}$ e $A_{01} = 1,772\times10^{-5}\ s^{-1}$.

Nosso modelo atômico inclui os cinco níveis de energia mais baixos: $2s^2\ 2p^4\ {}^3P_{2,1,0}$, $2s^2\ 2p^4\ {}^1D_2$ e $2s^2\ 2p^4\ {}^1S_0$. As energias foram tiradas de Moore (1993), e as probabilidades de transição foram extraídas do cálculo integrante do Iron Project de Galavís, Mendoza, & Zeippen (1997).

Como os níveis de estrutura fina do O^0 estão muito mais separados em comparação com os níveis de estrutura fina do C^0 e C^+, teremos que a radiação cósmica de fundo não desempenhará um papel relevante, como podemos ver pelas taxas de excitação na Tabela 2.1.

Os níveis excitados podem ser povoados por colisões com partículas presentes no meio. A Figura 2.4 mostra as taxas de excitação colisional dos níveis de estrutura fina do O^0 para vários parceiros de colisão. As taxas de excitação colisional por elétrons foram extraídas de Bell, Berrington, & Thomas (1998), por hidrogênio neutro de Launay & Roueff (1977) e por hélio neutro de Monteiro & Flower (1987). Para colisões com prótons empregamos os ajustes analíticos fornecidos por Péquignot (1990; 1996).

z	$B_{21}u_{21}$ [s^{-1}]	$B_{20}u_{20}$ [s^{-1}]
0	$3.0\ 10^{-41}$	$2.6\ 10^{-63}$
1	$4.0\ 10^{-23}$	$2.6\ 10^{-37}$
2	$4.4\ 10^{-17}$	$1.2\ 10^{-28}$
3	$4.6\ 10^{-14}$	$2.6\ 10^{-24}$
4	$3.0\ 10^{-12}$	$1.0\ 10^{-21}$
5	$4.8\ 10^{-11}$	$5.5\ 10^{-20}$

Tabela 2.1: Taxas de excitação dos níveis de estrutura fina do O^0 pela radiação cósmica de fundo.

Por motivos de completeza também consideramos excitação colisional dos níveis superiores $^1D_2$ e $^1S_0$. As forças de colisão médias para transições induzidas por elétrons envolvendo estes níveis foram tiradas de Berrington & Burke (1981). A taxa para a transição ^3P-^1D induzida por colisões com hidrogênio neutro foi obtida de Federman & Shipsey (1983). As taxas foram convertidas de acoplamento LS para as transições entre os níveis de estrutura fina individuais de acordo com os pesos estatísticos:

$$\begin{aligned} \gamma(^3P_J \rightarrow {}^1D_2) &= \frac{2J+1}{9}\gamma(^3P \rightarrow {}^1D) \ , \\ \gamma(^3P_J \rightarrow {}^1S_0) &= \frac{2J+1}{9}\gamma(^3P \rightarrow {}^1S) \ . \end{aligned} \tag{2.14}$$

Após incluir 135 transições permitidas ultravioleta envolvendo os níveis ^3P e níveis superiores do trabalho de Verner, Verner, & Ferland (1996), obtivemos as taxas de excitação indireta pelo campo de radiação da Galáxia (Gondhalekar, Phillips, & Wilson 1980): $\Gamma_{21} = 3{,}9\times10^{-11}\ s^{-1}$ e $\Gamma_{20} = 1.1\times10^{-11}\ s^{-1}$.

O povoamento dos níveis de estrutura fina do O^0 pode ser significativamente afetado por reações de troca de carga com hidrogênio (Péquignot 1990, 1996):

$$\begin{aligned} H^+ + O^0(^3P_J) &\rightarrow H^0(^2S_{1/2}) + O^+(^4S_{3/2}) \ , \\ H^0(^2S_{1/2}) + O^+(^4S_{3/2}) &\rightarrow H^+ + O^0(^3P_{J'}) \ . \end{aligned} \tag{2.15}$$

A inclusão deste processo exigiria o acoplamento do equilíbrio estatístico dos níveis com o equilíbrio de ionização, o que está fora do escopo deste trabalho. Portanto, em nossa análise consideramos apenas o caso em que o meio está primariamente neutro.

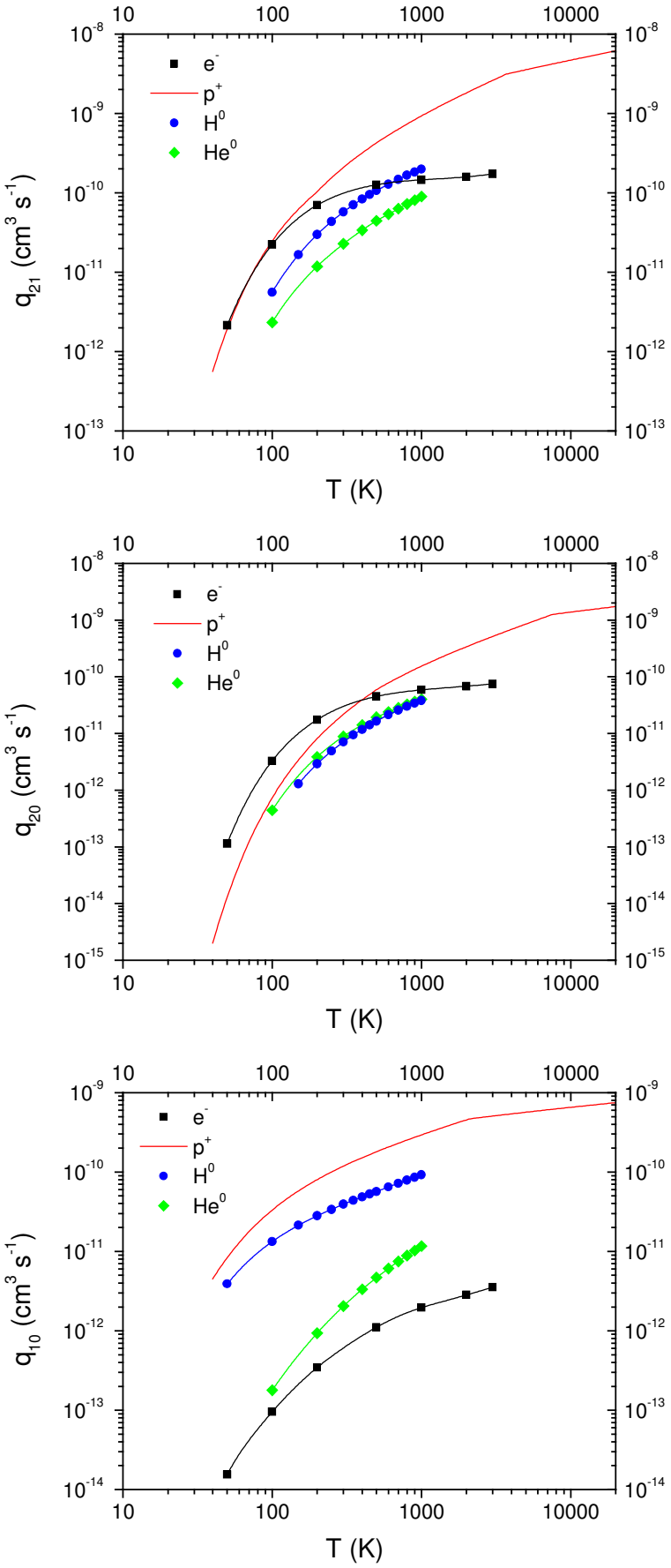


Figura 2.4: Taxas de excitação colisional para o O^0.

Taxas de excitação colisional $q_{JJ'}$ entre os níveis de estrutura fina $^3P_J \rightarrow {}^3P_{J'}$ do O^0 para vários parceiros de colisão.

Nas figuras 2.5 e 2.6 plotamos as razões de povoamento dos níveis de estrutura fina do O^0 sob várias condições físicas. Consideramos colisões com átomos de hidrogênio e hélio, assumindo uma abundância de hélio de 10% em relação ao hidrogênio (por número). Colisões com hélio aumentam o povoamento do primeiro nível de estrutura fina em relação ao estado fundamental por apenas 5%, e 10% para o segundo nível excitado (reduzindo a zero próximo ao equilíbrio termodinâmico no limite de altas densidades). As curvas que incluem o efeito da radiação cósmica de fundo no povoamento do primeiro estado excitado coincidem com as respectivas curvas que levam em consideração o campo ultravioleta da Galáxia; isto se deve ao fato de as taxas de excitação relevantes serem aproximadamente iguais $K_{21}^{z=5} \simeq \Gamma_{21}^{\mathrm{G}}$.

Nossos resultados não são diretamente comparáveis aos obtidos por Péquignot (1990; 1996), que fez considerações acerca do estado de ionização do gás. Porém, chamamos a atenção para a importância de se atualizar as taxas de excitação por elétrons empregada naquele trabalho – extraída de Berrington (1988) – para os cálculos mais recentes de Bell et al. (1998), uma vez que as taxas destes últimos são substancialmente mais baixas.

2.6. O íon Fe^+

O estado fundamental do Fe^+ é composto pelo sexteto $3d^6\ 4s\ {}^6D_{9/2,7/2,5/2,3/2,1/2}$. Se comparado aos outros átomos/íons analisados, o Fe^+ apresenta os níveis de estrutura fina muito mais separados uns dos outros e as probabilidades de transição são consideravelmente mais altas. Por exemplo, o primeiro nível excitado está situado em 384,790 cm^{-1} acima do nível fundamental, e a probabilidade de transição correspondente é $A_{7/2,9/2} = 2{,}13\times10^{-3}\ \mathrm{s}^{-1}$. Ambos os fatores contribuirão para que as razões de povoamento dos níveis de estrutura fina do Fe^+ sejam significativamente baixas.

Nosso modelo inclui os quatro termos LS mais baixos: $3d^6\ 4s\ {}^6D$, $3d^7\ {}^4F$, $3d^6\ 4s\ {}^4D$ e $3d^7\ {}^4P$, perfazendo um total de 16 níveis quando a estrutura fina é levada em consideração. As energias foram extraídas de Corliss & Sugar (1982), e as probabilidades de transição do cálculo integrante do Iron Project de Quinet, Le Dourneuf, & Zeippen (1996)[☞].

Em função da grande separação dos níveis de estrutura fina, a radiação cósmica de fundo não será um mecanismo de excitação importante. Por exemplo, mesmo em $z = 5$ a taxa de excitação para o primeiro nível excitado é apenas $B_{9/2,7/2}u_{9/2,7/2} = 3{,}5\times10^{-18}\ \mathrm{s}^{-1}$.

☞ Note que existe um pequeno erro para duas transições listadas na tabela 5 do artigo de Quinet et al. (1996). Os autores não adicionaram uma componente de dipolo magnético às probabilidades de transição, de modo que os valores deveriam ser: a ${}^4F_{7/2}$ – a ${}^4P_{5/2}$ = $8{,}83\times10^{-3}\ \mathrm{s}^{-1}$, a ${}^4F_{5/2}$ – a ${}^4P_{5/2}$ = $1{,}68\times10^{-3}\ \mathrm{s}^{-1}$, tal como aparece na tabela 4 deles.

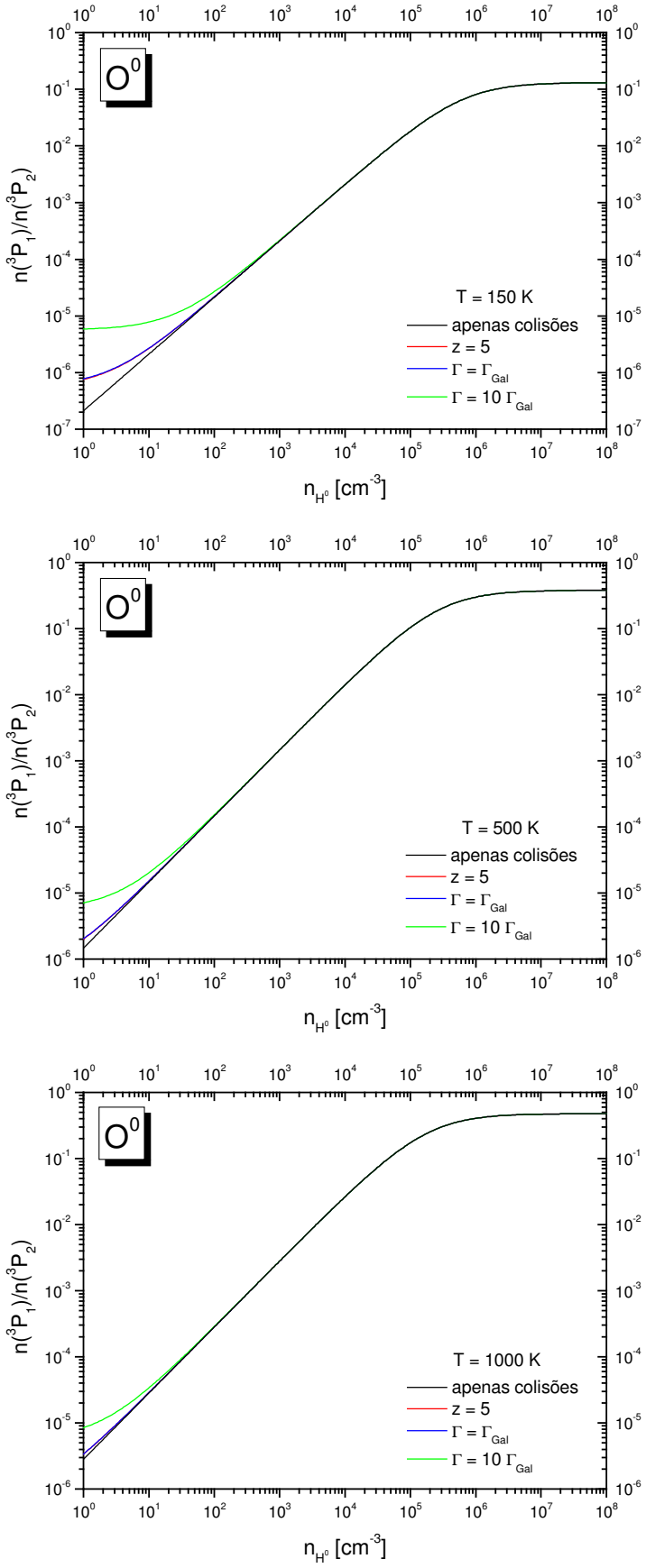


Figura 2.5: Povoamento relativo dos níveis de estrutura fina $n(^3P_1)/n(^3P_2)$ do O^0.

Povoamento do primeiro nível de estrutura fina em relação ao estado fundamental para o O^0. A curva correspondente a $z = 5$ coincide com a curva $\Gamma = \Gamma_{Gal}$.

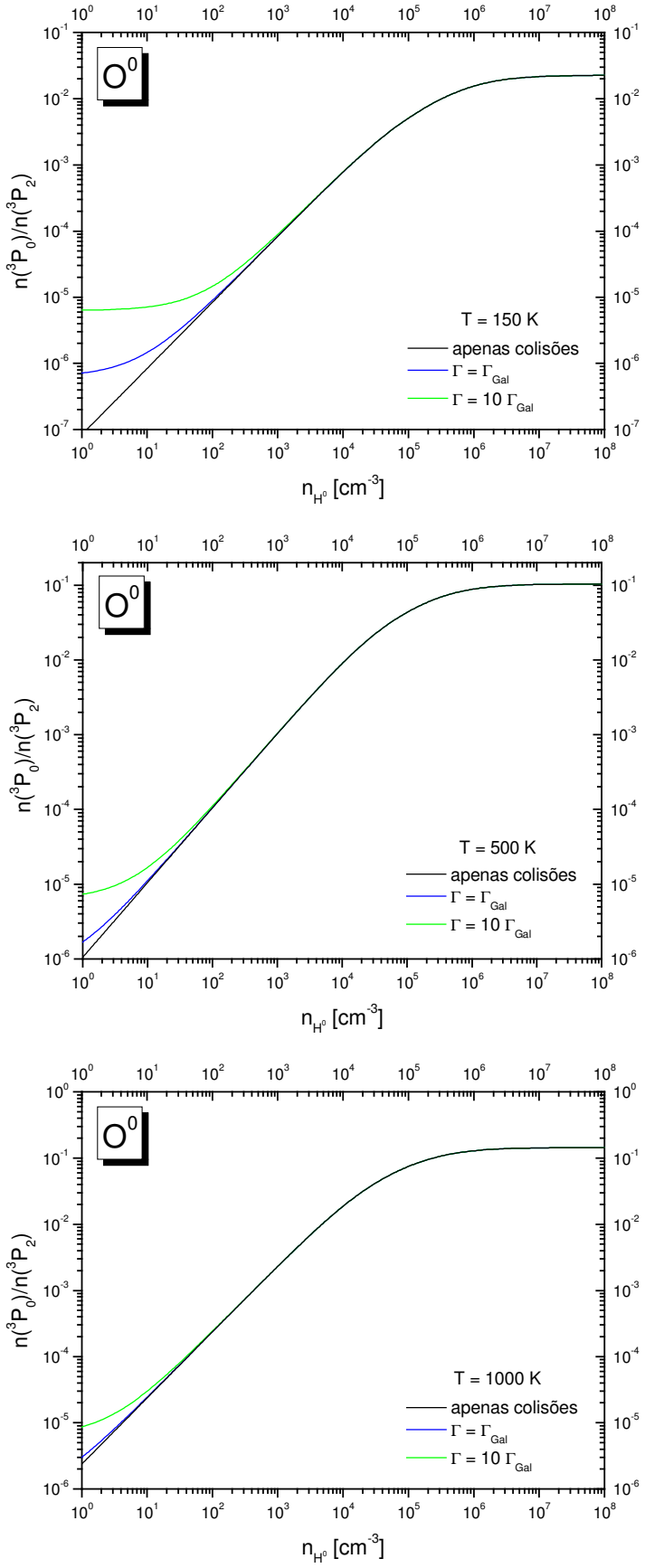


Figura 2.6: Povoamento relativo dos níveis de estrutura fina $n(^3P_0)/n(^3P_2)$ do O^0.

Povoamento do segundo nível de estrutura fina em relação ao estado fundamental para o O^0.

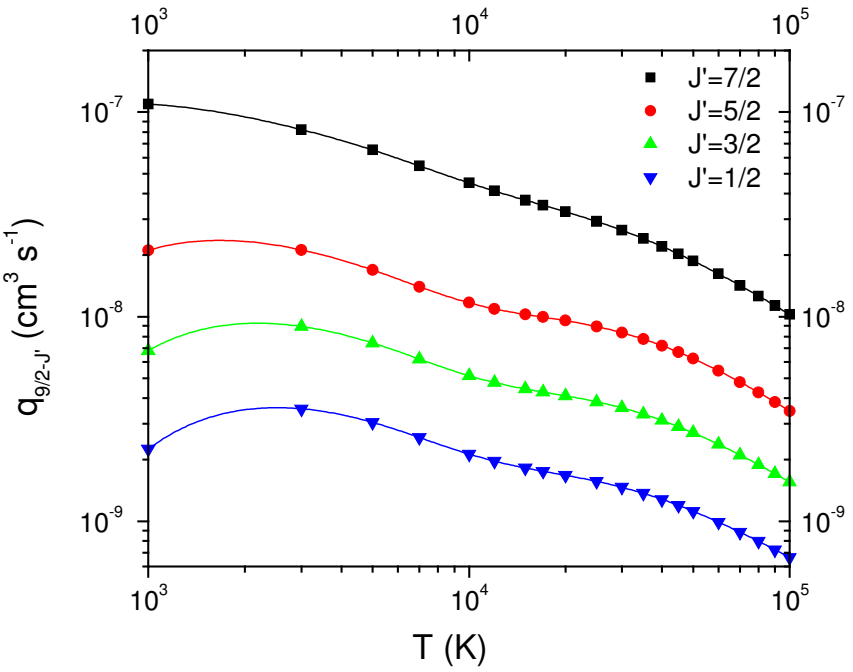


Figura 2.7: Taxas de excitação colisional para o Fe^+.
Taxas de excitação colisional por elétrons envolvendo o estado fundamental e os níveis de estrutura fina excitados $^6D_{9/2} \rightarrow {}^6D_{J'}$ do Fe^+.

O único processo colisional para o qual pudemos encontrar cálculos detalhados na literatura foram colisões com elétrons. A Figura 2.7 mostra as taxas de excitação devido a colisões com elétrons para as transições mais importantes dentro do termo fundamental ^6D. As forças de colisão médias correspondentes foram obtidas do cálculo integrante do Iron Project de Zhang & Pradhan (1995).

Nussbaumer & Storey (1980) estimaram as taxas de excitação por colisões com prótons como sendo menos de 10% das taxas de excitação por colisões com elétrons correspondentes para temperaturas até $T = 15.000$ K. Porém, como podemos notar nas figuras 2.2 e 2.3, as taxas de excitação para processos colisionais envolvendo íons positivos e prótons cresce rapidamente com a temperatura, de modo que devemos ter cautela ao negligenciar colisões com prótons a temperaturas extremamente altas.

Também incluímos em nosso modelo 212 transições permitidas envolvendo os níveis do termo fundamental ^6D e níveis superiores tiradas do cálculo integrante do Iron Project de Nahar (1995). As taxas de excitação indireta por fluorescência induzida pelo campo ultravioleta da Galáxia (Gondhalekar, et al. 1980) foram estimadas em $\Gamma_{9/2,7/2} = 7,0\times10^{-10}$ s^{-1}, $\Gamma_{9/2,5/2} = 1,3\times10^{-10}$ s^{-1} e $\Gamma_{9/2,3/2} = \Gamma_{9/2,1/2} = 0$ (estas últimas taxas são nulas porque as transições envolvidas não são permitidas por dipolo elétrico).

Cálculos anteriores das razões de povoamento dos níveis de estrutura fina do Fe^+ foram realizados por Keenan et al. (1988). Estes autores incluíram apenas os dois termos LS mais baixos

no modelo atômico deles, $3d^6\ 4s\ ^6D$ e $3d^7\ ^4F$, argumentando que os próximos dois termos LS, $3d^6\ 4s\ ^4D$ e $3d^7\ ^4P$, não afetariam significativamente o povoamento dos níveis do termo fundamental ^6D. Entretanto, cálculos teste mostraram que a inclusão dos dois termos LS seguintes aumentam o povoamento dos níveis do termo fundamental ^6D em até 17% para $T = 10.000$ K. Os testes consistiram em comparar os resultados obtidos empregando os modelos atômicos de 9 e 16 níveis, considerando apenas colisões com elétrons (sob várias densidades volumétricas) e decaimento espontâneo. Assim, concluímos que devemos empregar o modelo de 16 níveis para $T > 10.000$ K.

Com o intuito de avaliar a importância a altas temperaturas de níveis de energia ainda mais altos no povoamento dos níveis do termo fundamental ^6D, expandimos o nosso modelo atômico de 16 níveis de modo a englobar os dois termos LS seguintes: $3d^7\ ^2G$ e $3d^7\ ^2P$, com isso aumentando o total de níveis para 20.

Devido a limitações de espaço, Quinet et al. (1996) listam apenas probabilidades de transição para as transições mais fortes ($A_{ij} > 10^{-3}$ s^{-1}) envolvendo estes níveis. Assim, por motivo de completeza, decidimos complementar o trabalho deles com as transições mais fracas de Garstang (1962). As forças de colisão médias para estas transições foram extraídas do cálculo integrante do Iron Project de Bautista & Pradhan (1996).

Cálculos teste com o modelo atômico de 20 níveis revelaram que os últimos dois termos LS não afetam de modo significativo o povoamento dos níveis do termo fundamental ^6D até $T \simeq 20.000$ K, a temperatura mais alta considerada no cálculo de Bautista & Pradhan.

As figuras 2.8 e 2.9 mostram as razões de povoamento dos níveis de estrutura fina do Fe^+ como função da densidade eletrônica para $T = 1.000 - 20.000$ K. Podemos notar uma leve inversão no povoamento dos níveis $^6D_{3/2}$ e $^6D_{1/2}$ a baixas densidades.

Comparando nossos resultados com os derivados anteriormente por Keenan et al. (1988) sob as mesmas condições físicas, pudemos constatar que os povoamentos dos níveis de estrutura fina do termo fundamental ^6D que obtivemos são um fator de 2-4 mais altos. Acreditamos que esta discrepância se origina das forças de colisão médias utilizadas, uma vez que os valores de Zhang & Pradhan (1995) são muito mais altos do que os de Berrington et al. (1988), citados por Keenan et al. Uma vez que o trabalho de Zhang & Pradhan delineia a estrutura de ressonâncias das forças de colisão em detalhes, acreditamos que os resultados obtidos por nós devam ser mais confiáveis.

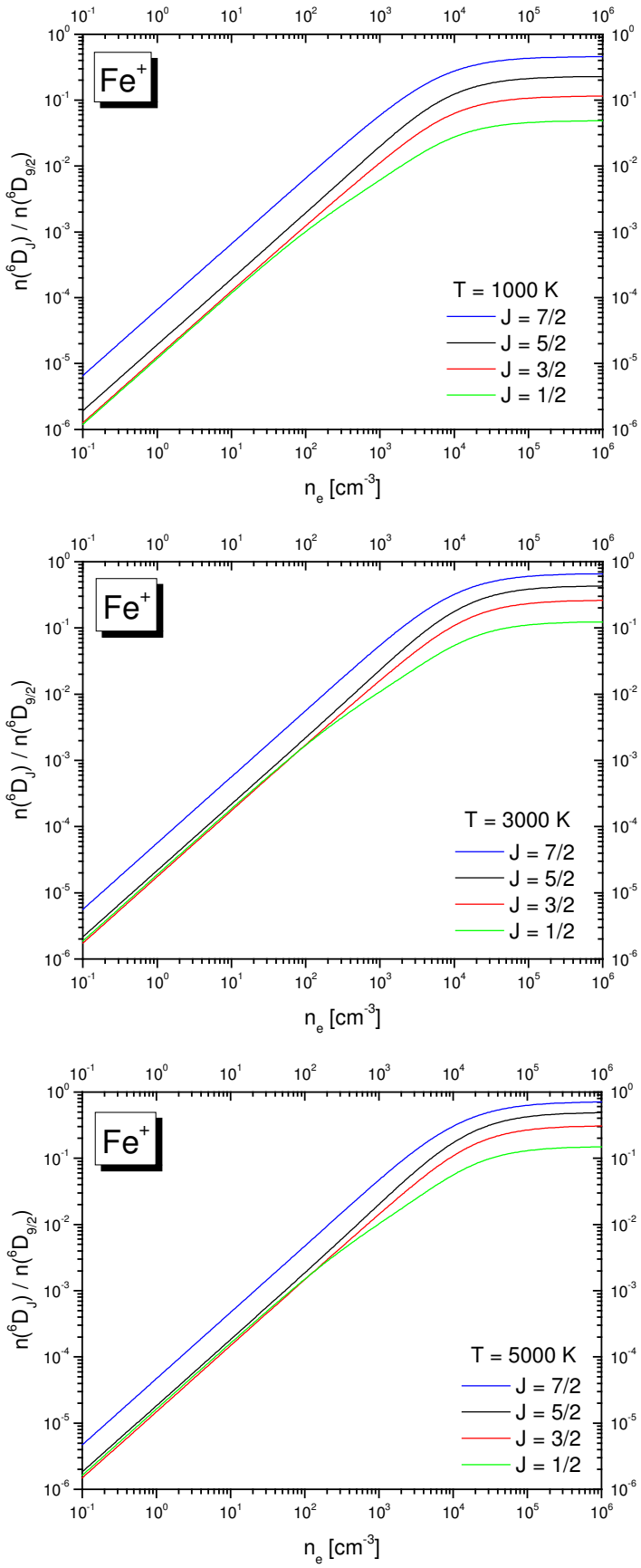


Figura 2.8: Povoamento relativo dos níveis de estrutura fina do Fe^+: T = 1.000 – 5.000 K .

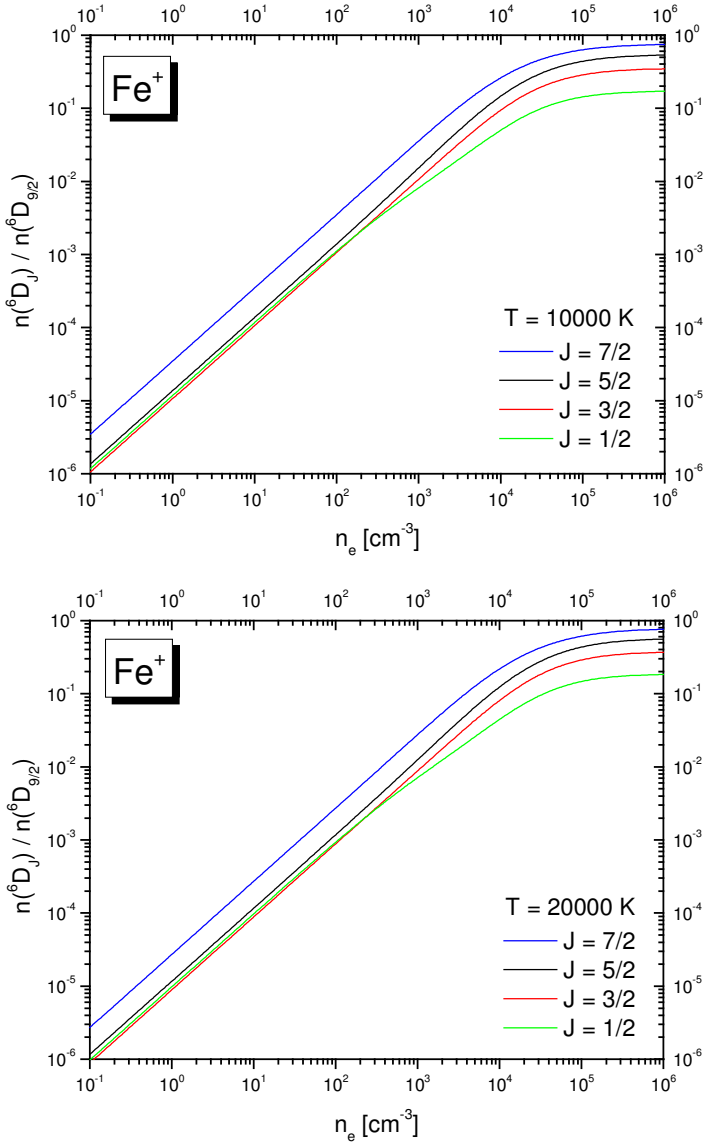


Figura 2.9: Povoamento relativo dos níveis de estrutura fina do Fe^+: T = 10.000 - 20.000 K .

Referências

Anderson, E., et al. 1999, *"LAPACK User's Guide"*, 3ª ed. (Philadelphia, PA: SIAM)

Bautista, M. A., & Pradhan, A. K. 1996, A&AS, 115, 551

Bell, K. L., Berrington, K. A., & Thomas, M. R. J. 1998, MNRAS, 293, L83

Berrington, K. A. 1988, J. Phys. B, 21, 1083

Berrington, K. A., & Burke, P. G. 1981, Planet. Space Sci., 29, 377

Berrington, K. A., Burke, P. G., Hibbert, A., Mohan, M., & Baluja, K. L. 1988, J. Phys. B, 21, 339

Blackford, L. S., Dongarra, J. J., Du Croz, J., Hammarling, S., & Wasniewski, J. 1996, *"LAPACK working note 117: a Fortran 90 interface for LAPACK."*

Corliss, C., & Sugar, J. 1982, JPCRD, 11, 135

De Boor, C. 1978, *"A Practical Guide to Splines"* (Berlin/Heidelberg: Springer)

Dongarra, J., Du Croz, J., Duff, I., & Hammarling, S. 1990, ACM Trans. Math. Software, 16, 1

Dongarra, J. J., Du Croz, J., Hammarling, S., & Hanson, R. J. 1988, ACM Trans. Math. Software, 14, 1

Federman, S. R., & Shipsey, E. J. 1983, ApJ, 269, 791

Fixsen, D. J., Cheng, E. S., Gales, J. M., Mather, J. C., Shafer, R. A., & Wright, E. L. 1996, ApJ, 473, 576

Galavís, M. E., Mendoza, C., & Zeippen, C. J. 1997, A&AS, 123, 159

Garstang, R. H. 1962, MNRAS, 124, 321

Gondhalekar, P. M., Phillips, A. P., & Wilson, R. 1980, A&A, 85, 272

Hamann, F. W., Barlow, T. A., Chaffee, F. C., Foltz, C. B., & Weymann, R. J. 2001, ApJ, 550

Keenan, F. P., Hibbert, A., Burke, P. G., & Berrington, K. A. 1988, ApJ, 332, 539

Kolb, E. W., & Turner, M. S. 1990, *"The Early Universe"* (Nova Iorque: Addison-Wesley)

Kool, M., et al. 2001, ApJ, 548, 609

Launay, J. M., & Roueff, E. 1977, A&A, 56, 289

Lima, J. A. S., Silva, A. I., & Viegas, S. M. 2000, MNRAS, 312, 747

Monteiro, T. S., & Flower, D. R. 1987, MNRAS, 228, 101

Moore, C. E. 1993, in *"CRC Handbook of Chemistry and Physics"*, ed. J. W. Gallagher, 76ª ed. (Boca Raton, FL: CRC Press), p. 336

Nahar, S. N. 1995, A&A, 293, 967

Netlib, Repositório de bibliotecas numéricas de código aberto: http://www.netlib.org

Nussbaumer, H., & Storey, P. J. 1980, A&A, 89, 308

Osterbrock, D. E. 1989, *"Astrophysics of Gaseous Nebulae and Active Galactic Nuclei"* (Mill Valley, CA: University Science Books)

Péquignot, D. 1990, A&A, 231, 499

---. 1996, A&A, 313, 1026 (errata)

Quinet, P., Le Dourneuf, M., & Zeippen, C. J. 1996, A&AS, 120, 361

Rybicki, G. B., & Lightman, A. P. 1979, *"Radiative Processes in Astrophysics"* (Nova Iorque: John Wiley & Sons)

Silva, A. I. 1999, dissertação de mestrado, *"Linhas de estrutura fina em absorção no espectro de QSOs."* (Instituto Astronômico e Geofísico, Universidade de São Paulo, São Paulo) http://www.ignacioalex.com/bibliografia

Silva, A. I., & Viegas, S. M. 2001, Comp. Phys. Comm., 136, 319

---. 2002, MNRAS, 329, 135

Smoot, G. F., & Scott, D. 1998, Eur. Phys. J. C, 3, 127

Srianand, R., & Petitjean, P. 2000, A&A, 357, 414

---. 2001, A&A, 373, 816

Verner, D. A., Verner, E. M., & Ferland, G. J. 1996, ADNDT, 64, 1

Wampler, E. J., Chugai, N. N., & Petitjean, P. 1995, ApJ, 443, 586

Zhang, H. L., & Pradhan, A. K. 1995, A&A, 293, 953

3. O Banco de Dados Atômicos

3.1. Introdução

Simulações numéricas de nebulosas gasosas fotoionizadas constituem uma ferramenta importantíssima para analisar e entender o espectro de uma infinidade de objetos, tais como regiões H II, nebulosas planetárias, remanescentes de supernovas, núcleos ativos de galáxias, o meio intergaláctico, etc. Se considerarmos ainda que grande parte da informação que temos acerca do universo é extraída de fótons, podemos ter uma idéia da abrangência dos problemas nos quais este instrumental é empregado em astronomia.

Atualmente existem cerca de uma meia dúzia de códigos de fotoionização espalhados pelo mundo, cujos autores se reúnem com uma certa periodicidade para compará-los em detalhes e discutir as causas das possíveis discrepâncias. Os resultados do último destes encontros são apresentados por Péquignot et al. (2001).

Basicamente o que estes códigos fazem é tomar a radiação de uma fonte qualquer e estudar como ela é absorvida e reprocessada por uma nebulosa, simulando uma gama de processos atômicos que ocorrem no interior da mesma. A estratégia consiste em dividir a nuvem em regiões espaciais finitas onde se supõe as condições físicas como constantes, e então calcular o transporte de radiação entre essas regiões. Como a maioria dos códigos existentes ainda implementa modelos unidimensionais, teremos que as regiões de condições físicas constantes serão camadas esféricas concêntricas (ou placas plano-paralelas) de uma certa espessura e situadas a uma certa distância da fonte de radiação. Dentro de cada uma dessas camadas é suposto o equilíbrio de ionização, que considera todos os processos que ionizam e recombinam o gás, além do equilíbrio térmico, que avalia todos os processos que o aquecem e o resfriam. A teoria das nebulosas gasosas fotoionizadas está descrita em pormenores no texto de Osterbrock (1989), ou ainda em Mihalas (1978) para aspectos mais relacionados ao transporte de radiação.

Embutidos nos códigos de fotoionização, há um grande número de parâmetros atômicos, tais como seções de choque de fotoionização, taxas de recombinação, probabilidades de transição, forças de colisão médias, etc. Teremos então que a confiabilidade dos resultados obtidos em nossas simulações numéricas será fortemente influenciada pela qualidade do banco de dados atômicos utilizado.

Nas últimas décadas houve um crescimento vertiginoso - tanto em quantidade quanto em qualidade – no volume de dados atômicos disponíveis na literatura, impulsionado sobretudo pelo aumento do poder computacional disponível. Grandes colaborações internacionais, tais como o Opacity Project (Seaton et al. 1994) e o Iron Project (Hummer et al. 1993), foram estabelecidas com intuito de produzir dados em larga escala e preencher as lacunas existentes no banco de dados atômicos atual.

Foi com o intuito de incorporar alguns destes avanços recentes ao código de fotoionização Aangaba que decidimos fazer uma revisão do seu conjunto de dados atômicos, antes de utilizá-lo em nossa aplicação aos sistemas em absorção dos QSOs. Esta é uma tarefa árdua não apenas pela enorme quantidade de parâmetros necessários, mas sobretudo pelo fato de os dados se encontrarem espalhados por inúmeras fontes nas publicações da astronomia e da física. Assim, para completá-la em tempo hábil procuramos nos ater ao seguinte objetivo: reunir a quantidade de dados atômicos necessários para possibilitar o cálculo do equilíbrio de ionização de todos os elementos do H ao Zn. Com isso limitamos nossa busca a seções de choque de fotoionização, taxas de recombinação, probabilidades de ejeção de elétrons Auger, taxas de troca de carga com hidrogênio e hélio e taxas de ionização colisional para todos os 465 íons dos elementos do H ao Zn.

A etapa de levantamento das taxas de troca de carga e ionização colisional foi concluída com sucesso, e as fontes de onde foram extraídos os dados são discutidas em detalhes no capítulo 4. Infelizmente não foi possível reunir um conjunto apropriado de seções de choque de fotoionização e coeficientes de recombinação, pelo fato de a literatura conter inúmeras brechas. Porém pudemos obter resultados interessantes e inéditos, os quais expomos no capítulo 5 juntamente com as dificuldades encontradas nesta etapa.

Neste capítulo nos ocuparemos de considerações gerais sobre o banco de dados atômicos e também descrevemos o formato utilizado nos arquivos de dados atômicos.

Todo o banco de dados atômicos está disponível no CD que acompanha a tese.

3.2. Aspectos gerais

Ao projetarmos o nosso banco de dados atômicos, BDA, nos preocupamos em atender a três requisitos essenciais:

1) Sua estrutura deve possibilitar a alteração dos dados atômicos de modo fácil e imediato;

Esta é uma necessidade imperativa, uma vez que novos dados continuarão inundando a literatura num ritmo cada vez mais acelerado. Embutir os dados atômicos em subrotinas não é uma solução satisfatória, pois é extremamente dispendioso retestá-las a cada nova alteração. É preciso

isolar o código do BDA para protegê-lo de mudanças, permitindo que ambos possam evoluir de maneira independente.

2) A documentação das fontes de onde foram extraídos os dados deve estar embutida dentro do BDA em si;

Muitas vezes a documentação acerca da origem dos dados atômicos sequer é motivo de preocupação dos usuários. Incluir a documentação junto com os dados dificulta o seu extravio e facilita a tarefa de mantê-la atualizada. No futuro, a única maneira de saber se um novo cálculo é mais confiável do que aquele que está implementado no BDA é nos referirmos ao artigo original.

3) Os dados devem ser incorporados de maneira facilmente legível ao ser humano, e na forma mais próxima possível das fontes originais.

Com isto estamos elevando a preocupação em incorporar os dados de forma correta e livre de erros acima de qualquer outra. Não devemos neste ponto nos focar em criar estruturas de dados que possam ser eficientemente lidas pela máquina, uma vez que manipular grandes quantidades de dados é uma tarefa propensa a erros humanos. E ao entrarmos os dados na forma mais próxima possível das fontes originais estamos facilitando a checagem de erros e também sua inclusão imediata, por dispensar manipulações e/ou conversões adicionais (temos aqui, portanto, uma sobreposição com o primeiro requisito acima).

A solução desenvolvida consistiu em implementar o BDA como uma série de arquivos HTML, os quais são posteriormente "compilados" para arquivos binários que podem ser lidos de maneira rápida e eficiente pela máquina. Por incluírem formatação de texto (negrito, itálico, cores, etc.), figuras e *hiperlinks*, arquivos HTML são extremamente legíveis ao ser humano. Como o HTML é um padrão bem estabelecido, praticamente todas as plataformas contam com vários navegadores que podem ser utilizados para visualizar os dados prontamente. Editá-los também não apresenta maiores dificuldades, já que arquivos HTML nada mais são do que arquivos texto comuns envolvidos por *tags* de formatação[19].

Na seção seguinte introduzimos a sintaxe dos arquivos de dados atômicos e veremos como o formato escolhido preenche os requisitos acima.

---

[19] O usuário também pode recorrer a inúmeros editores visuais disponíveis no mercado, alguns inclusive gratuitos.

3.3. A sintaxe dos arquivos de dados atômicos

Cada um dos arquivos de dados atômicos é composto por uma coleção de blocos de dados, sendo que o início de cada bloco é demarcado pela *string* `[DATA_BLOCK]`. Quaisquer comentários inseridos antes desta *string* são ignorados, e portanto o usuário está livre para preceder o início de cada bloco com um cabeçalho documentando a fonte dos dados.

O primeiro bloco serve apenas para especificar o número de blocos de dados no arquivo, por exemplo:

```
[DATA_BLOCK]
10
```

Tipicamente o cabeçalho correspondente a este primeiro bloco especificará a qual processo atômico os dados se referem, qual a cobertura dos mesmos (*i.e.*, para quais íons são entradas as taxas) e conterá também um breve histórico de revisão do arquivo.

Nos blocos seguintes com os dados atômicos, o usuário poderá especificá-los de duas formas principais: como uma tabela de pontos (para interpolação linear) ou como um ajuste analítico. Para entrar os dados como uma tabela, a sintaxe é:

```
[DATA_BLOCK]
2
table
3
C+3
1.0E4    2.0E4    5.0E4
5.0E-9   8.7E-9   1.2E-8
2
SiIV
1.0E4     1.0E5
1.0E-15   1.0E-15
```

O primeiro número especifica o número de entradas no bloco (neste exemplo entramos dois íons), após o qual segue-se a palavra-chave `table`. Para cada uma das entradas que se seguem devemos então especificar na primeira linha o número de pontos na tabela, na linha seguinte o íon ao qual nos referimos, e nas duas linhas seguintes as abscissas e ordenadas dos pontos, respectivamente.

Se os dados estão expressos na forma de ajustes, a sintaxe a ser obedecida é:

```
íon     Xmin      Xmax       C1       C2     C3      C4
[DATA_BLOCK]
2
fit
3   4
Fe+1  1.0E4   1.0E5   2.3E-9   0.3   2.0   -3.8
Ca7   1.0E4   1.0E5   4.9E-9   1.2   0.5   -3.7
```

Na primeira linha temos o número de entradas, na segunda a palavra-chave `fit` e na terceira especificamos o número da fórmula de ajuste que estamos empregando e quantos parâmetros ela contém. O usuário poderá optar por uma dentre 24 fórmulas de ajustes pré-definidas, sumarizadas no apêndice A. Estas 24 fórmulas foram introduzidas para suportar os dados reunidos na versão atual do BDA; é possível criar novas fórmulas de ajustes com facilidade, como veremos no capítulo 8. No exemplo acima estamos empregando a fórmula de ajuste número três, a qual possui quatro parâmetros. Cada uma das entradas ocupa uma linha, onde especificamos em seqüência o nome do íon, o intervalo de validade do ajuste e os coeficientes da fórmula.

Como podemos notar nos exemplos acima, é possível especificar os nomes dos íons seguindo duas notações: de acordo com o estágio de ionização ou com a carga do íon. Em astronomia costuma-se seguir a primeira convenção, onde o átomo neutro é o primeiro estágio de ionização, o íon uma vez ionizado é o segundo, e assim por diante. Na sintaxe do BDA, o estágio de ionização pode ser entrado em números arábicos ou romanos, *e.g.*, `SiIV`, `Ca7`, etc. Para adotar a segunda convenção, devemos colocar o sinal `+` entre o símbolo do elemento e a sua carga: `C+3`, `Fe+1`, etc., a não ser pelo átomo neutro que pode ser entrado tanto como `Ar+0` quanto como `Ar0` (pois neste caso não há ambigüidade com a primeira notação). Para os íons uma vez ionizados devemos entrar `C+1`, ao invés de apenas `C+`. Os símbolos dos elementos precisam necessariamente ser entrados com o primeiro caractere em maiúsculo e o segundo, se houver, em minúsculo; números romanos devem sempre figurar em maiúsculas.

Os exemplos acima ilustram a forma básica de entrar os dados das tabelas ou ajustes, que é dispondo-os ao longo de linhas. Porém, de acordo com os nossos dados, pode ser mais prático organizá-los de maneiras diferentes. Suponhamos, por exemplo, que desejamos entrar uma tabela com vários íons cujos pontos foram calculados nas mesmas abscissas; neste caso para evitar repetições podemos adotar a sintaxe:

```
[DATA_BLOCK]
3
table   commonrow
3
        1.0E4    2.0E4    5.0E4
LiI     1.7E-8   2.1E-8   5.0E-9
LiII    9.2E-8   9.9E-8   1.0E-7
LiIII   4.8E-9   7.2E-9   3.6E-9
```

Sendo que agora empregamos também a palavra-chave `commonrow`; há ainda outras palavras-chave que possibilitam diferentes formas de entrar os dados, como mostra a Tabela 3.1 . Caso o usuário não especifique nenhuma delas é assumido como padrão o modo `row`.

tabela	palavra-chave	ajuste
npontos atomo/ion x1 x2 x3 ... y1 y2 y3	row	atomo/ion xmin xmax c1 c2 c3 ... ...
npontos atomo/ion x1 y1 x2 y2 x3 y3	column	atomo/ion xmin xmax c1 c2 c3 ... ...
npontos npontos ... atomo/ion atomo/ion ... x1 y1 x1 y1 ... x2 y2 x2 y2 ... x3 y3 x3 y3	multicolumn	atomo/ion atomo/ion ... xmin xmin ... xmax xmax ... c1 c1 ... c2 c2 ... c3 c3 ... ...
npontos x1 x2 x3 ... atomo/ion y1 y2 y3 ... atomo/ion y1 y2 y3	commonrow	Não se aplica.
npontos atomo/ion atomo/ion ... x1 y1 y1 ... x2 y2 y2 ... x3 y3 y3	commoncolumn	Não se aplica.

Tabela 3.1: Possíveis formas de se organizar os dados no BDA.

As palavras-chave `linear_linear`, `log_log`, `linear_log`, `log_`, `_linear` e variantes servem para indicar se os valores entrados na verdade correspondem aos logarítmos das abscissas ou ordenadas. Por exemplo, a palavra-chave `log_` denota que estamos entrando com os logarítmos das abscissas. O padrão aqui é `linear_linear`. Analogamente palavras-chave do tipo `eV_` e `_Mb` indicam as unidades em que estão expressas as abscissas ou ordenadas, sendo que os valores padrão correspondem às unidades no sistema CGS. Estas palavras-chave só se aplicam quando estamos entrando os dados na forma de tabelas; no caso dos ajustes devemos embutir os fatores de conversão de unidades e logarítmos dentro das respectivas fórmulas.

As palavras-chave `Z_ne` e `Z_st` servem para expressar os íons na forma de dois números inteiros, ao invés da *string* com o formato acima. Ambos os números devem aparecer consecutivamente na mesma linha, o primeiro se referindo à carga do núcleo e o segundo pode se referir ao número de elétrons ou ao estágio de ionização, conforme o caso.

Por fim, a palavra-chave `shell` se aplica apenas a processos de ionização que conduzem à ejeção de elétrons Auger. Neste caso devemos fornecer um campo a mais nos dados contendo uma *string* especificando a partir de qual camada se dá a ionização. Esta *string* deverá se repetir no arquivo contendo as probabilidades de ejeção de elétrons Auger a partir daquela camada para o íon em questão.

Condensando toda a informação acima, teremos então que o formato geral para os arquivos de dados atômicos obedecerá à sintaxe:

```
"[DATA_BLOCK]"
numero_de_entradas_neste_bloco
table/fit [ROW/column/multicolumn/commonrow/commoncolumn] ■
        ■ [LINEAR_LINEAR/linear_log/log_/_linear/...] [eV_/_Mb] ■ (se tabela)
        ■ [Z_ne/Z_st] [shell]
numero_da_formula_de_ajuste  numero_de_parametros  (se ajuste)
```

Palavras-chave entre colchetes são opcionais, sendo que o valor padrão em cada caso aparece em maiúsculas. O símbolo ■ denota continuação na mesma linha. Poderá haver qualquer número de espaços em branco separando as palavras-chave, entretanto elas deverão ocorrer na ordem especificada. Nenhuma palavra-chave diferencia caracteres maiúsculos/minúsculos, exceto `[DATA_BLOCK]`, que deve ser entrada exatamente desta forma.

Caso o usuário deseje incluir *tags* HTML ele deverá cuidar para que nenhuma dessas *tags* invada as seções de blocos de dados, para que não haja nenhuma interferência durante a compilação dos arquivos. Outro cuidado que se faz necessário é envolver todo o bloco de dados pela *tag* `<pre>`, a fim de que, ao visualizar os dados, o navegador não reformate estas linhas:

```
<pre>
[DATA_BLOCK]
...
</pre>
```

No apêndice A anexamos como exemplo um arquivo completo de dados atômicos.

3.4. Benefícios e problemas

O esquema proposto para o BDA é ao mesmo tempo simples e flexível, e, além de preencher todos os requisitos apontados, ele proporciona também outros benefícios interessantes:

1) Testes de sanidade;

Ao invés de simplesmente converter os arquivos HTML para arquivos binários, a implementação em *software* do BDA pode também se encarregar de aplicar alguns testes simples

para tentar apanhar erros grosseiros nos dados. Taxas negativas, por exemplo, não fazem sentido algum, e provavelmente são o resultado de erros ao entrar os parâmetros de algum ajuste.

2) Testes de completeza;

Também é possível verificar se não há lacunas no conjunto de dados atômicos fornecido. Por exemplo, se desejamos calcular o equilíbrio de ionização de algum elemento, então é preciso certificar-se de que existe pelo menos um processo de ionização e outro de recombinação para cada um dos íons deste elemento (do contrário o sistema linear a ser resolvido será indeterminado!).

Note que não é conveniente alertar para repetições nos dados, pois ao entrarmos duas taxas para o mesmo processo podemos na verdade estar nos referindo a duas componentes distintas. Este é o caso, por exemplo, dos coeficientes de recombinação, os quais em geral são separados em três contribuições: radiativa, dieletrônica a baixas temperaturas e dieletrônica a altas temperaturas. Assim, quando forem encontradas várias entradas para o mesmo processo devemos combiná-las numa "função composta", que adiciona todas elas.

3) Otimização dos dados;

Já vimos que a estrutura do BDA prioriza facilitar a entrada dos dados de maneira correta e o mais próxima possível das fontes originais. Ao converter os dados para serem manipulados pela máquina, podemos otimizá-los de forma que as taxas sejam calculadas mais rapidamente. Por exemplo, suponhamos que um dado autor se encarregou de detalhar alguma taxa com 100 pontos; porém, pode ser possível reproduzí-la dentro da incerteza do cálculo com apenas 5 pontos, o que resultará numa interpolação mais rápida. Outras otimizações poderiam procurar minimizar a quantidade de memória requerida, se o volume de dados é considerável.

4) Fácil alteração de um BDA para outro.

O produto final do processo de compilação dos arquivos de dados atômicos é um único arquivo binário contendo todo o BDA. Entretanto o usuário poderá desejar obter não apenas um, mas sim vários BDAs com conjuntos de dados atômicos diferentes. Em suas aplicações ele poderá então prontamente avaliar a dependência dos seus resultados com a combinação de dados atômicos utilizada.

Do outro lado desta mesma moeda, vários autores poderiam produzir um único arquivo de dados atômicos de referência com o propósito de melhor comparar as previsões de seus códigos, ou

seja, dessa vez anulando as diferenças causadas pelo conjunto de dados atômicos. Este BDA de referência não deverá necessariamente incluir os dados atômicos mais confiáveis, mas sim aqueles que proporcionam maior estabilidade aos cálculos. Embora também seja possível criar um BDA padrão com os melhores dados atômicos disponíveis, provavelmente os inúmeros autores discordarão em algumas escolhas e optarão por utilizar os seus próprios conjuntos de dados favoritos.

Todos estes benefícios resultaram de nossos esforços em conceber um formato para o BDA que satisfizesse aos requisitos discutidos anteriormente da maneira mais simples e flexível possível. Em nenhum momento nos preocupamos com a estrutura em *software* por detrás do BDA que se encarregaria de transformá-lo em realidade, e por isso deixamos escapar uma dificuldade embaraçosa na qual viemos a esbarrar mais tarde.

De acordo com o paradigma da programação estruturada (a maneira pela qual estamos habituados a programar), um problema complexo é resolvido dividindo-o em tarefas pequenas, as quais podem ainda ser subdivididas em tarefas menores e assim sucessivamente até o decompormos em partes simples que possam ser facilmente atacadas.

Seguindo esta abordagem, a maneira natural de implementar o BDA em *software* seria criar uma série de subrotinas que devolveriam as taxas como função da temperatura para um dado íon; por exemplo, para os coeficientes de recombinação teríamos algo semelhante a (em Fortran):

```
function recomb(Z,N,T)
real :: recomb  ! taxa de recombinação [cm^3 s^-1]
integer, intent(in) :: Z  ! número atômico
integer, intent(in) :: N  ! estágio de ionização
real, intent(in) :: T  ! temperatura [K]

.........................................

end function recomb
```

No corpo da rotina embutiríamos ainda os dados necessários para o cálculo do coeficiente de recombinação; por exemplo, se estes são dados na forma de um ajuste com cinco parâmetros, teríamos então um *array* Z x N x 5 com os coeficientes dos ajustes para todos os íons do BDA. Porém, a taxa para certos íons pode ser expressa como um ajuste, para outros como uma tabela, e existe ainda a possibilidade de a expressarmos como uma combinação de ajustes e tabelas. Determinados íons podem ser também ajustados por fórmulas diferentes, cada qual com um número distinto de parâmetros. Assim, para refletir mudanças nos dados atômicos, quase sempre seríamos forçados a também alterar a rotina.

Evidentemente esta não é uma dificuldade incontornável, apenas queremos salientar que provavelmente não é possível implementar toda a flexibilidade de nosso BDA de maneira trivial empregando programação estruturada.

Para criar a implementação em *software* do BDA preferimos utilizar ferramentas modernas de programação, às quais introduzimos o leitor resumidamente no capítulo 6. Estas ferramentas permitiram a resolução do problema acima de maneira natural (como veremos no capítulo 8) além de proporcionar inúmeras outras vantagens.

Referências

Hummer, D. G., Berrington, K. A., Eissner, W., Pradhan, A. K., Saraph, H. E., & Tully, J. A. 1993, A&A, 279, 298

Mihalas, D. 1978, *"Stellar Atmospheres"*, 2ª ed. (São Francisco: Freeman)

Osterbrock, D. E. 1989, *"Astrophysics of Gaseous Nebulae and Active Galactic Nuclei"* (Mill Valley, CA: University Science Books)

Péquignot, D., et al. 2001, in *"Spectroscopic Challenges of Photoionized Plasmas"*, eds. G. Ferland, & D. W. Savin (Lexington: ASP Conference Series, vol. 247), p. 533

Seaton, M. J., Yan, Y., Mihalas, D., & Pradhan, A. K. 1994, MNRAS, 266, 805

4. Troca de Carga e Ionização Colisional

4.1. Troca de carga

Reações de troca de carga são muito importantes para o equilíbrio de ionização de certos elementos, e precisam ser levadas em consideração para explicar o espectro observado de nebulosas fotoionizadas (Péquignot, Aldrovandi, & Stasinska 1978). Estas reações se dão da seguinte forma:

$$\mathrm{X}^{+i} + \mathrm{Y}^{0} \rightarrow \mathrm{X}^{+(i-1)} + \mathrm{Y}^{+} + \Delta E \ , \tag{4.1}$$

onde ΔE é o defeito de energia, e os parceiros de troca de carga mais importantes são Y = H, He, por serem geralmente mais abundantes na nuvem. Reações que se processam no sentido indicado pela seta acima são denominadas recombinação por troca de carga, ao passo que o processo inverso é chamado ionização por troca de carga.

Devido à energética dos processos, as reações de recombinação por troca de carga são tipicamente mais importantes por serem exotérmicas ($\Delta E > 0$). Na aproximação nebular, teremos então o cenário em que um dado íon num estágio de ionização qualquer e inicialmente povoado no estado fundamental, após a colisão é levado ao estágio de ionização anterior podendo em geral ser deixado também num nível de energia excitado. Se o nível fundamental do íon resultante for um canal eficiente para a ocorrência da reação, teremos então que a reação inversa também será importante e as respectivas taxas dos processos estarão relacionadas pelo princípio do balanceamento detalhado:

$$\frac{\delta'}{\delta} = \frac{g}{g'} e^{-\frac{\Delta E}{kT}} \ , \tag{4.2}$$

onde δ' é a taxa de ionização por troca de carga, δ é a taxa de recombinação, g é o peso estatístico dos reagentes e g' dos produtos (de acordo com a reação (4.1)).

Mesmo quando uma reação não apresenta nenhum canal eficiente, ela pode ainda ocorrer radiativamente:

$$X^{+i} + Y^0 \rightarrow X^{+(i-1)} + Y^+ + \gamma \ . \tag{4.3}$$

A taxa de reação nestes casos é da ordem de 10^{-14} cm^3 s^{-1}, independente da temperatura.

Para a atualização das reações de troca de carga com hidrogênio, o ponto de partida foi o trabalho de Kingdon & Ferland (1996, daqui em diante KF). Nele os autores fizeram uma compilação das taxas de troca de carga para todos os elementos do H ao Zn envolvendo os estágios de ionização com carga q = 1 – 4. As eventuais brechas na literatura, em geral para íons de baixa abundância astrofísica, foram complementadas por novos cálculos na aproximação de Landau-Zener. Subseqüentemente à publicação do trabalho, os autores se preocuparam em manter atualizações constantes de algumas taxas de troca de carga que foram surgindo na literatura desde então. Essas atualizações atualmente se encontram disponíveis no banco de dados de troca de carga na WWW (BDTC).

Uma atualização importante que não consta nesse banco de dados são as taxas de troca de carga com o oxigênio, calculadas por Stancil et al. (1999b):

$$O^+(^4S) + H^0 \rightleftarrows O^0(^3P_J) + H^+ \ . \tag{4.4}$$

Devido aos dois íons possuírem um potencial de ionização muito próximo um do outro, a reação é quase ressonante e as taxas serão elevadas. Uma vez que o oxigênio é um dos principais resfriadores do gás, a reação acima será importante não apenas no equilíbrio de ionização do oxigênio, mas também no equilíbrio térmico da nuvem.

No cálculo de Stancil et al. (1999b) foram empregados vários métodos diferentes (cada qual válido num dado regime de energias), levando-se em consideração inclusive dados experimentais, para produzir uma curva recomendada para os coeficientes de troca de carga como função da temperatura. Em $T = 10^4$ K, uma temperatura característica de nebulosas fotoionizadas, os novos dados são um fator 2 maiores do que os coeficientes de Field & Steigman (1971), citados na compilação de KF.

Para a reação de ionização por troca de carga acima, assumimos que todos os átomos de oxigênio neutro estão povoados no nível de estrutura fina fundamental J = 2, de acordo com a aproximação nebular. Uma possível melhoria seria futuramente considerar reações de troca de carga a partir dos níveis excitados de estrutura fina, incluindo-os explicitamente no equilíbrio de ionização do O^0 - O^+ .

Pudemos verificar ainda um problema com a compilação de KF no que diz respeito à reação:

$$He^{+} + H^{0} \rightarrow He^{0} + H^{+} \ . \qquad (4.5)$$

A referência que os autores dão para esta reação é o trabalho de Zygelman et al. (1989), que fornece apenas taxas de troca de carga radiativas numa faixa de temperaturas diferente do range de validade do ajuste fornecido por KF. A Figura 4.1 ilustra a discrepância e compara o ajuste de KF com a taxa de recombinação direta obtida aplicando o princípio de balanceamento detalhado às taxas da reação inversa a (4.5), calculadas por Kimura et al. (1993b). Felizmente os coeficientes de troca de carga são demasiadamente pequenos para desempenhar algum papel no equilíbrio de ionização do hélio.

Concluindo, os coeficientes de troca de carga com hidrogênio adotados foram extraídos de KF, incluindo-se as atualizações disponíveis na WWW (aqui sumarizadas na Tabela 4.1) e também os novos resultados de Stancil et al. (1999b) para o oxigênio.

Para as reações de troca de carga com hélio, uma compilação freqüentemente citada na literatura é o trabalho de Arnaud & Rothenflug (1985, daqui em diante AR), que publicaram ajustes aos dados calculados por Butler, Heil & Dalgarno (1980) e Butler & Dalgarno (1980) - daqui em diante coletivamente referidos como BHD. Como a compilação de AR não abrange tantas referências como na compilação de KF, preferimos neste caso adotar as fontes originais em BHD.

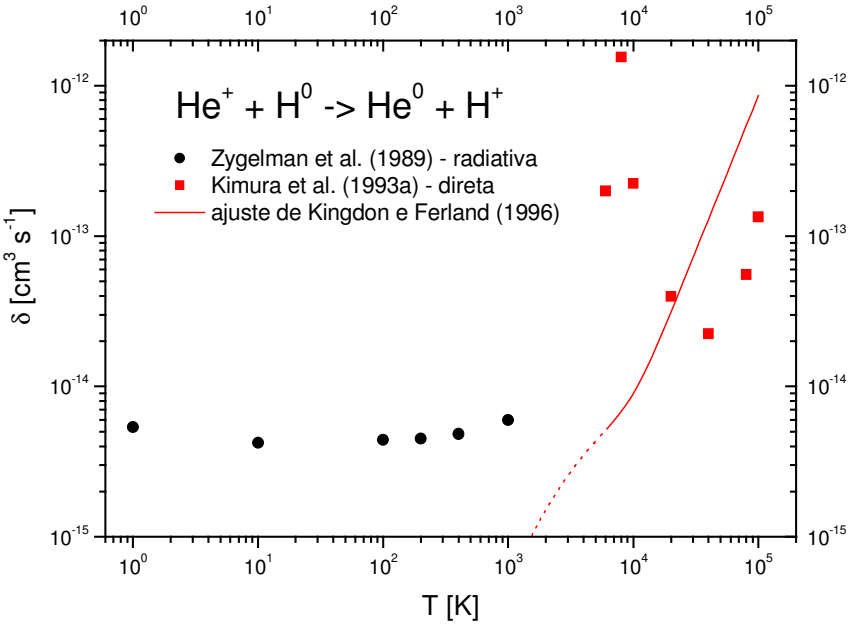


Figura 4.1: Taxas de ionização por troca de carga com hidrogênio para o He^{+}.

X^{+i}	recomb./ion.	comentário
Li^0	ionização	resultado original revisto
B^{+4}	recombinação	resultado original revisto
C^0	ionização	novo ajuste baseado no cálculo de Stancil et al. (1998)
C^+	recombinação	novo ajuste baseado no cálculo de Stancil et al. (1998)
N^{+4}	recombinação	resultado original revisto
Na^0	ionização	novo ajuste baseado no cálculo de Croft & Dickinson (1996)
Si^0	ionização	novo ajuste baseado no cálculo de Kimura et al. (1996b)
Si^+	ionização	novo ajuste baseado no cálculo de Clarke et al. (1998)
Si^{+2}	recombinação	novo ajuste baseado no cálculo de Clarke et al. (1998)
Fe^0	ionização	nova taxa estimada em 4/5 do coeficiente de Langevin

Tabela 4.1: Atualizações às taxas de troca de carga com hidrogênio de Kingdon & Ferland (1996).
Extraído de (BDTC).

BHD calcularam taxas de recombinação por troca de carga para os íons C^{+3}-C^{+4}, N^{+2}-N^{+4}, O^{+2}-O^{+4}, Ne^{+2}-Ne^{+4}, Mg^{+3}-Mg^{+4}, Si^{+3}-Si^{+4}, S^{+3}-S^{+4}, Ar^{+2}-Ar^{+4} e ionização por troca de carga para os íons C^+, N^+, Si^+-Si^{+2}, S^+-S^{+2}, Ar^+.

Em AR, o defeito de energia para a reação de ionização do N^+ está erroneamente colocado 1.44 eV (vide a tabela IIIB do artigo deles), ao invés de 5.03 eV como aparece originalmente em Butler & Dalgarno (1980, tabela 2). Por outro lado, o aparente erro de AR ao colocar o defeito de energia para a reação de ionização do Si^{+2} em 8.88 eV contra 6.88 eV em BHD, está na verdade correto e trata-se de um erro do dado original em Butler e Dalgarno (1980)! Cabe lembrar que as taxas das reações de ionização por troca de carga são extremamente sensíveis ao valor adotado para o defeito de energia, por este entrar dentro de um fator exponencial no ajuste analítico, como na equação (4.2). A Figura 4.2 ilustra esse fato para a reação de ionização do Si^{+2} e compara os resultados com os novos cálculos de Stancil et al. (1999a).

Algumas atualizações aos resultados de BHD que pudemos encontrar na literatura estão listados na Tabela 4.2 . Na Figura 4.3 comparamos as taxas de recombinação por troca de carga do Si^{+4} calculadas por Stancil et al. (1997) com os resultados anteriores de Opradolce, McCarroll & Valiron (1985) e Butler & Dalgarno (1980).

Novos coeficientes de troca de carga com hélio estão sendo calculados na aproximação de Landau-Zener e disponibilizados no banco de dados de troca de carga mencionado acima. Como este banco de dados ainda se encontra nos estágios iniciais de desenvolvimento, tais coeficientes serão incorporados tão logo os novos resultados estejam concluídos e publicados.

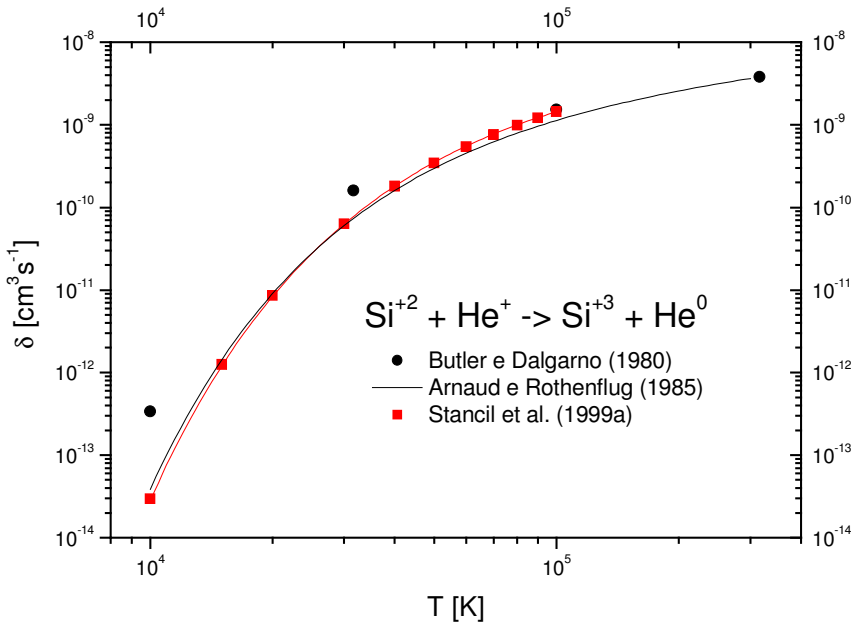


Figura 4.2: Taxas de ionização por troca de carga com hélio para o Si^{+2}.
O ajuste de Arnaud & Rothenflug (1985) é baseado no cálculo de Butler & Dalgarno (1980), porém corrigido um valor errôneo para o defeito de energia da reação empregado pelos últimos. Os pontos e a curva vermelha correspondem ao cálculo mais recente de Stancil et al. (1999a).

Em suma, para os coeficientes de troca de carga com hélio empregamos o conjunto de dados de BHD, complementado pelas atualizações descritas na Tabela 4.2 .

Por fim, consideramos ainda reações de troca de carga com hidrogênio e hélio com espécies altamente ionizadas com cargas $q > 4$. Devido ao fato desses íons não existirem em abundância nas mesmas regiões onde temos o hidrogênio e o hélio na fase neutra, tais reações não são geralmente relevantes para o equilíbrio de ionização. Porém, devido à enorme diferença de potencial entre os reagentes, vários estados estão energeticamente acessíveis para a reação e as taxas de troca de carga serão então consideravelmente altas. Em algumas situações este processo mostrou-se significativo no equilíbrio de ionização do ferro (Ferland et al. 1997).

X^{+i}	recomb./ion.	fonte
C^{0}	ionização	Kimura et al. (1993a)
O^{+2}	recombinação	Kimura et al. (1996a)
Si^{+2}	ionização	Stancil et al. (1999a)
Si^{+3}	recombinação	Stancil et al. (1999a)
Si^{+4}	recombinação	Stancil et al. (1997)

Tabela 4.2: Referências para reações de troca de carga com hélio publicadas na literatura desde Butler & Dalgarno (1980) e Butler et al. (1980).

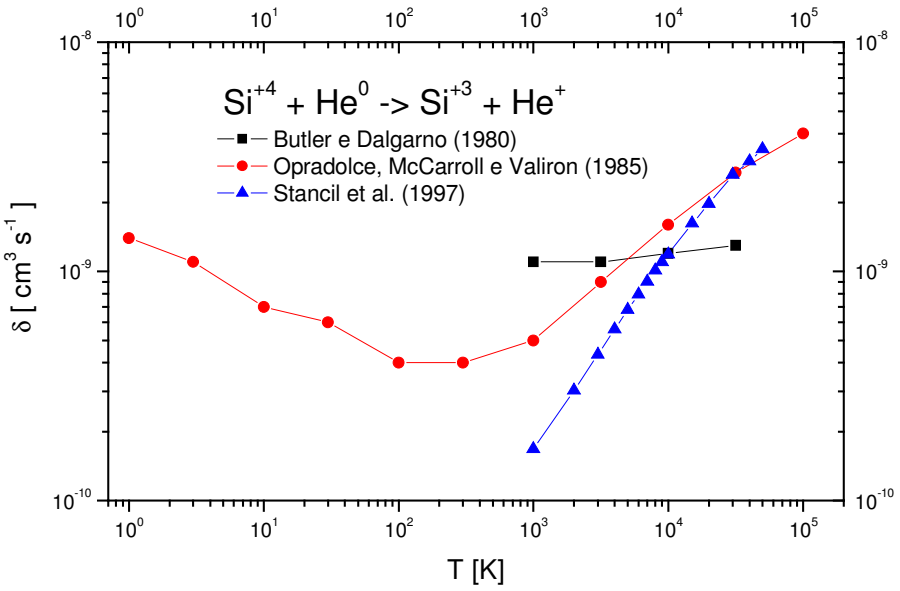


Figura 4.3: Taxas de recombinação por troca de carga com hélio para o Si^{+4}.

Se assumirmos que toda colisão que consegue vencer a barreira centrífuga leva a uma reação de troca de carga, teremos a taxa de Langevin (Ferland, et al. 1997):

$$\delta = 3.85\times10^{-8}\, z(\alpha\, m_e / \mu)^{1/2}\ \mathrm{cm}^3\ \mathrm{s}^{-1}\ , \tag{4.6}$$

onde z é a carga efetiva do íon, α é a polarizabilidade de dipolo e μ / m_e é a massa reduzida expressa em termos da massa do elétron m_e.

Para íons do Fe colidindo com hidrogênio teremos

$$\delta = 1.92\times10^{-9} z\ \ \mathrm{cm}^3\ \mathrm{s}^{-1}\ , \tag{4.7}$$

e com hélio

$$\delta = 0.54\times10^{-9} z\ \ \mathrm{cm}^3\ \mathrm{s}^{-1}\ . \tag{4.8}$$

Para reações de recombinação por troca de carga com hidrogênio para os íons Al^{+5} e Ca^{+5} empregamos as taxas de Kingdon, Ferland & Feibelman (1995).

4.2. Ionização colisional

A ionização colisional se dá quando o átomo ou íon colide com elétrons livres do meio. Basicamente esse processo pode ocorrer por duas rotas principais: *ionização direta* ou *excitação-autoionização*. No primeiro caso o elétron da camada de valência é arrancado diretamente pela colisão com o elétron livre, ao passo que no segundo caso a colisão com o elétron livre apenas cria um buraco na camada interna do átomo ou íon, que, por se tornar altamente excitado, acaba se autoionizando expontaneamente:

$$\begin{array}{ll} \mathrm{X}^{+i} + e^- \rightarrow \mathrm{X}^{+i+1} + e^- + e^- & \text{ionização direta} \\ \mathrm{X}^{+i} + e^- \rightarrow \mathrm{X}^{+i**} + e^- \rightarrow \mathrm{X}^{+i+1} + e^- + e^- & \text{excitação-autoionização} \end{array} . \tag{4.9}$$

Em geral, a excitação-autoionização só será significativa em altas temperaturas e para íons que possuem poucos elétrons na camada de valência e inúmeros elétrons na camada imediatamente mais interna, como os íons da seqüência isoeletrônica do Li e Na (estes últimos contam com apenas um elétron na camada externa 3s e seis elétrons na camada 2p imediatamente abaixo).

Para os coeficientes de excitação colisional o nosso banco de dados atômicos cobre todos os íons de todos os elementos do H ao Ge, e baseia-se essencialmente em dois grandes conjuntos de dados disponíveis na literatura e que passamos a discutir em seguida.

A primeira compilação deve-se a Arnaud et al. e cobre os elementos de abundância astrofísica: H, He, C, N, O, Ne, Na, Mg, Al, Si, S, Ar, Ca, Fe e Ni (Arnaud & Rothenflug 1985), sendo que os dados referentes aos íons do Fe foram posteriormente revisados (Arnaud & Raymond 1992).

Arnaud et al. ajustam as seções de choque de ionização colisional por uma fórmula motivada teoricamente e proposta originalmente por Younger (1981):

$$\sigma_{\mathrm{col}} = \sum_j \frac{1}{uI_j^2}\left[A_j\left(1-\frac{1}{u}\right)+B_j\left(1-\frac{1}{u}\right)^2+C_j\ln(u)+D_j\frac{\ln(u)}{u}\right], \tag{4.10}$$

onde $u = E/I_j$ e a soma é realizada sobre todas as subcamadas do íon inicial. I_j é o potencial de ionização da camada j e A_j, B_j, C_j e D_j são parâmetros. C_j é o coeficiente de Bethe, e determina o comportamento da seção de choque a altas energias; ele pode ser ajustado ou calculado a partir da seção de choque de fotoionização σ_{ph}:

$$C = \frac{I}{\pi\alpha}\int_{I}^{\infty}\frac{\sigma_{\mathrm{ph}}(E)}{E}dE \ , \tag{4.11}$$

onde α é a constante de estrutura fina.

Convoluindo (4.10) com uma distribuição Maxwelliana de temperaturas obtemos para o coeficiente de ionização colisional

$$\begin{aligned} q_{\mathrm{col}} &= \frac{6.69\times10^{-7}}{(kT)^{3/2}}\sum_{j}\frac{e^{-x_j}}{x_j}F(x_j) \quad \mathrm{cm^3s^{-1}} \\ x_j &= \frac{I_j}{kT} \\ F(x_j) &= A_j[1-x_j f_1(x_j)]+B_j[1+x_j-x_j(2+x_j)f_1(x_j)]+C_j f_1(x_j)+D_j x_j f_2(x_j) \end{aligned} \tag{4.12}$$

onde kT e I_j são expressos em eV e A_j, B_j, C_j e D_j são dados em 10^{-14} cm^2 eV^2; $f_1(x)$ e $f_2(x)$ são as funções especiais:

$$\begin{aligned} f_1(x) &= e^x\int_1^{\infty}\frac{e^{-xt}}{t}dt \\ f_2(x) &= e^x\int_1^{\infty}\frac{\ln t}{t}e^{-xt}dt \end{aligned} \tag{4.13}$$

Fórmulas simples para avaliar $f_1(x)$ e $f_2(x)$ são reunidas no apêndice B.

O segundo conjunto de dados ao qual nos referimos acima é conhecido na literatura como os dados do "grupo de Belfast". Nele os autores compilaram as taxas de ionização colisional para todos os elementos do H até o O (Bell et al. 1983), e posteriormente estenderam o trabalho até o Ni (Lennon et al. 1988).

O grupo de Belfast empregou fórmulas consideravelmente mais simples do que a escolhida por Arnaud et al.:

$$\begin{aligned} q_{\mathrm{col}} &= e^{-\frac{I}{kT}}\sqrt{\frac{kT}{I}}\sum_{n=0}^{5}a_n\left[\log\frac{kT}{I}\right]^n && \frac{I}{10}\le kT\le 10I \\ q_{\mathrm{col}} &= \sqrt{\frac{I}{kT}}\left[\alpha\ln\left(\frac{kT}{I}\right)+\sum_{n=0}^{2}\beta_n\left(\frac{I}{kT}\right)^n\right] && kT>10I \end{aligned} \tag{4.14}$$

onde a_n, α e β_n são parâmetros.

Outros trabalhos freqüentemente citados na literatura astronômica são as compilações de Shull & van Steenberg (1982), que não se baseia num levantamento extenso de medidas com em Arnaud et al., e Landini & Monsignori Fossi (1990; 1991), que simplesmente extrapolaram os dados de Shull & van Steenberg para outros elementos menos abundantes.

Voronov (1997) reajustou os dados do grupo de Belfast com uma fórmula mais simples

$$q_{\mathrm{col}} = A\frac{1+P\sqrt{U}}{X+U}U^{K}e^{-U} \quad , \tag{4.15}$$

onde $U = dE/T$, dE é o potencial de ionização e A, X e K são parâmetros. O parâmetro P foi incluído para melhor ajustar as taxas de ionização de alguns íons próximo ao limiar, e pode ser apenas 0 ou 1.

Apesar de envolver menos parâmetros do que a fórmula (4.14) proposta pelo grupo de Belfast, a incerteza adicional introduzida ao se reajustar os dados (ainda que pequena para a maioria dos íons) não se justifica, uma vez que a relação (4.15) também envolve funções transcendentais e portanto não é significativamente mais barata do que a fórmula original do ponto de vista computacional.

Para nos guiar na escolha entre os dados de Arnaud et al. e do grupo de Belfast, nos valemos do excelente trabalho de Kato, Masai, & Arnaud (1991), que comparam detalhadamente ambas as compilações.

Kato et al. chamam a atenção para o fato de as taxas de alguns íons dos dados do grupo de Belfast necessitarem ser corrigidos por um fator 1.626, devido a um erro de impressão na tabela de Younger (1982). Voronov (1997) menciona ter aplicado esse fator de correção, o que é contestado por Mazzotta et al. (1998), que checaram os ajustes de Voronov e para alguns íons encontraram um acordo com os dados originais do grupo de Belfast sem esse fator de correção.

Uma diferença entre os trabalhos de Arnaud et al. e do grupo de Belfast é o fato de apenas os primeiros levarem em consideração excitação-autoionização, o que pode afetar significativamente as taxas de algumas seqüências isoeletrônicas como a do Li, Na e Mg a S (cada vez menos à medida que o número de elétrons p aumenta).

A baixas temperaturas os dados de Arnaud et al. são mais confiáveis em relação aos do grupo de Belfast. Isso se deve ao fato de os primeiros terem empregado uma fórmula mais apropriada para descrever o comportamento das seções de choque próximas ao limiar, e ainda devido ao fato do grupo de Belfast ter adotado para alguns íons cálculos teóricos baseados na aproximação de ondas planas de Born, que não é válida em baixas energias e tipicamente fornece

seções de choque mais baixas do que o método de ondas distorcidas por uma fator 2 próximo ao limiar.

Kato et al. também contestam o método empregado pelo grupo de Belfast para extrapolar algumas seções de choque e preencher as lacunas nos dados. Arnaud et al. extrapolam as seções de choque para cada subcamada individualmente, ao passo que o grupo de Belfast extrapolou diretamente as seções de choque totais.

Pelas razões descritas acima, decidimos adotar a compilação de Arnaud et al. sempre que possível, complementando os dados com os do grupo de Belfast apenas para os íons de menor abundância astrofísica não cobertos no trabalho de Arnaud et al. Para os dados de Arnaud et al. empregamos também os ajustes separados para excitação-autoionização para algumas seqüências isoeletrônicas que esses autores fornecem (basicamente Li, Na e Mg a S). O fator de correção apontado por Kato et al. para alguns íons nos dados do grupo de Belfast também foi levado em consideração. Alguns poucos íons em Arnaud et al. tiveram os dados ajustados incorretamente e por isso foram extraídos do grupo de Belfast: Na^{+} Al^{+3} Si^{+4} S^{+6} Ar^{+3}.

Certos íons não são cobertos nem nos trabalhos de Arnaud et al. nem nos do grupo de Belfast, e foram extraídos de Voronov (1997): V^{+15}-V^{+22}, Cr^{+23}, Mn^{+24} e Co^{+18}-Co^{+26}. Da mesma forma, os elementos Cu, Zn, Ga e Ge não foram considerados nestas compilações e foram obtidos de Mazzitelli & Mattioli (2002).

A Figura 4.4 ilustra as taxas de excitação colisional para os íons do carbono. A pequena descontinuidade em baixas temperaturas para a curva referente ao C^{0} aparece também em alguns outros íons cujas taxas foram tiradas da compilação de Arnaud et al.. Esta descontinuidade se origina das aproximações empregadas para calcular as funções especiais discutidas no apêndice B, cada qual válida dentro de um certo intervalo.

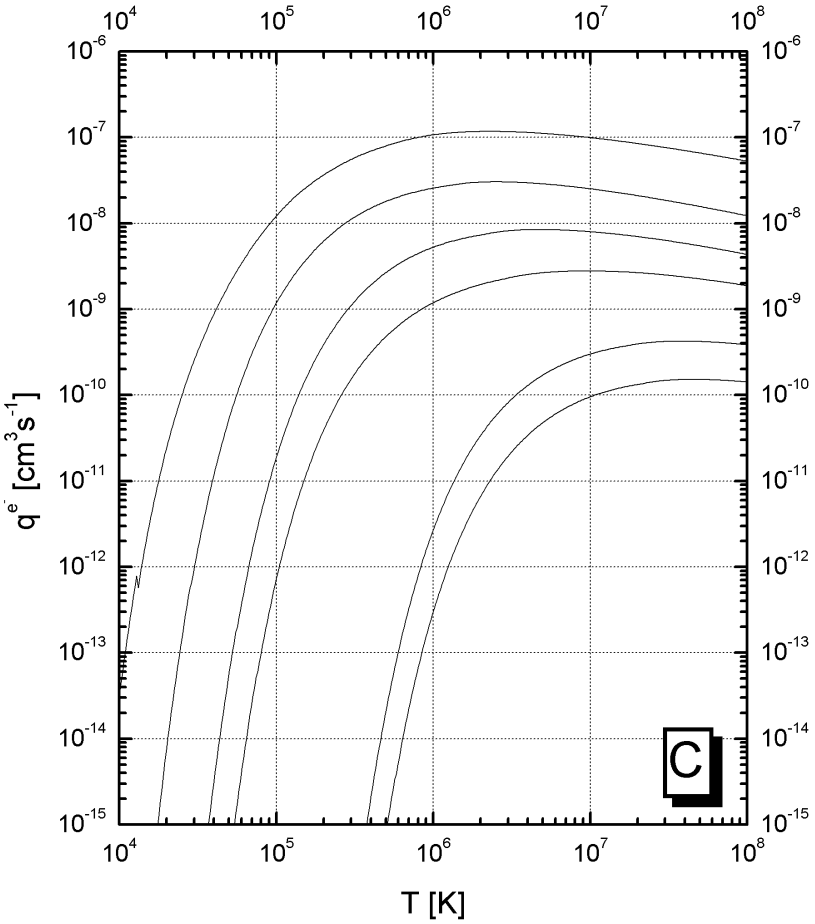


Figura 4.4: Taxas de ionização colisional para os íons do C.
As curvas seguem a ordem dos íons menos ionizados para os mais ionizados, de cima para baixo.

Referências

Arnaud, M., & Raymond, J. 1992, ApJ, 398, 394

Arnaud, M., & Rothenflug, R. 1985, A&AS, 60, 425

BDTC, Banco de dados de reações de troca de carga no Oak Ridge National Laboratory e University of Georgia: http://www-cfadc.phy.ornl.gov/astro/ps/data/

Bell, K. L., Gilbody, H. B., Hughes, J. G., Kingston, A. E., & Smith, F. J. 1983, JPCRD, 12, 891

Butler, S. E., & Dalgarno, A. 1980, ApJ, 241, 838

Butler, S. E., Heil, T. G., & Dalgarno, A. 1980, ApJ, 241, 442

Clarke, N. J., Stancil, P. C., Zygelman, B., & Cooper, D. L. 1998, J. Phys. B, 31, 533

Croft, H., & Dickinson, A. S. 1996, J. Phys. B, 29, 57

Ferland, G. J., Korista, K. T., Verner, D. A., & Dalgarno, A. 1997, ApJ, 481, L115

Field, G. B., & Steigman, G. 1971, ApJ, 166, 59

Kato, T., Masai, K., & Arnaud, M. 1991, *"Comparison of Ionization Rate Coefficients of Ions from Hydrogen through Nickel"*, NIFS-DATA-14 (National Institute for Fusion Science)

Kimura, M., Dalgarno, A., Chantranupong, L., Li, Y., Hirsch, G., & Buenker, R. J. 1993a, ApJ, 417, 812

Kimura, M., Gu, J. P., Hirsch, G., & Buenker, R. J. 1996a, Phys. Rev. A, 53, 4164

Kimura, M., Lane, N. F., Dalgarno, A., & Dixson, R. G. 1993b, ApJ, 405, 801

Kimura, M., Sannigrahi, A. B., Gu, J. P., Hirsch, G., Buenker, R. J., & Shimamura, I. 1996b, ApJ, 473, 1114

Kingdon, J., Ferland, G. J., & Feibelman, W. A. 1995, ApJ, 439, 793

Kingdon, J. B., & Ferland, G. J. 1996, ApJS, 106, 205

Landini, M., & Monsignori Fossi, B. C. 1990, A&AS, 82, 229

---. 1991, A&AS, 91, 183

Lennon, M. A., Bell, K. L., Gilbody, H. B., Hughes, J. G., Kingston, A. E., Murray, M. J., & Smith, F. J. 1988, JPCRD, 17, 1285

Mazzitelli, G., & Mattioli, M. 2002, ADNDT, 82, 313

Mazzotta, P., Mazzitelli, G., Colafrancesco, S., & Vittorio, N. 1998, A&AS, 133, 403

Opradolce, L., McCarroll, R., & Valiron, P. 1985, A&A, 148, 229

Péquignot, D., Aldrovandi, S. M. V., & Stasinska, G. 1978, A&A, 63, 313

Shull, J. M., & van Steenberg, M. 1982, ApJS, 48, 95 (veja errata em ApJS 49, 351)

Stancil, P. C., Clarke, N. J., Zygelman, B., & Cooper, D. L. 1999a, J. Phys. B, 32, 1523

Stancil, P. C., et al. 1998, ApJ, 502, 1006

Stancil, P. C., Schultz, D. R., Kimura, M., Gu, J. P., Hirsch, G., & Buenker, R. J. 1999b, A&AS, 140, 225

Stancil, P. C., Zygelman, B., Clarke, N. J., & Cooper, D. L. 1997, Phys. Rev. A, 55, 1064

Voronov, G. S. 1997, ADNDT, 65, 1

Younger, S. M. 1981, J. Quant. Spectrosc. Radiat. Transfer, 26, 329

---. 1982, J. Quant. Spectrosc. Radiat. Transfer, 27, 541

Zygelman, B., Dalgarno, A., Kimura, M., & Lane, N. F. 1989, Phys. Rev. A, 40, 2340

5. Fotoionização/Recombinação

5.1. Introdução

Fotoionização e recombinação direta em geral estão entre os processos mais importantes que ocorrem em plasmas astrofísicos expostos a uma fonte de radiação. No limite em que estes dois processos dominam sobre os demais mecanismos de ionização/recombinação, teremos que o equilíbrio de ionização de um dado elemento X será dado por:

$$n(\mathrm{X}^{+i})\int_{\nu_0}^{\infty}\frac{4\pi J_\nu}{h\nu}\sigma_{\mathrm{f}}^{\mathrm{fot}}(\nu)d\nu = n_{\mathrm{e}}n(\mathrm{X}^{+i+1})\alpha_{\mathrm{Tot}}(T) \quad , \tag{5.1}$$

onde $n(\mathrm{X}^{+i})$ é a densidade volumétrica de íons do elemento X no estágio de ionização i, J_ν é a intensidade média do campo de radiação, ν é a freqüência do fóton ionizante (a partir de uma freqüência limite ν_0), $\sigma_{\mathrm{f}}^{\mathrm{fot}}$ é a seção de choque de fotoionização a partir do estado fundamental, n_{e} é a densidade eletrônica e $\alpha_{\mathrm{Tot}}(T)$ é o coeficiente de recombinação total como função da temperatura cinética do gás.

Na equação acima supusemos como válida a *aproximação nebular*, segundo a qual a maior parte dos íons se encontrarão povoados unicamente no estado fundamental. Assim, o número de fotoionizações a partir do estado fundamental será balanceado pelo número de recombinações do estágio de ionização seguinte, também a partir do estado fundamental porém podendo levar a qualquer um dos estados do íon resultante.

Seções de choque de fotoionização e recombinação estão relacionadas pelo princípio do balanceamento detalhado (Osterbrock 1989):

$$\sigma_{\mathrm{f}}^{\mathrm{rec}}(v) = \frac{g_i}{g_{i+1}}\frac{h^2\nu^2}{m^2c^2v^2}\sigma_{\mathrm{f}}^{\mathrm{fot}}(\nu) \tag{5.2}$$

onde h é a constante de Planck, m é a massa do elétron, c é a velocidade da luz, g_i é o peso estatístico do estágio de ionização i e $\sigma_{\mathrm{f}}^{\mathrm{rec}}$ é a seção de choque de recombinação como função da velocidade do elétron fotoejetado v:

$$\frac{1}{2}mv^2 + h\nu_0 = h\nu \quad . \tag{5.3}$$

O coeficiente de recombinação para o estado fundamental é obtido convoluindo-se a seção de choque com uma distribuição Maxwelliana de velocidades $f(v)$:

$$\alpha_{\mathrm{f}}(T) = \left\langle v\sigma_{\mathrm{f}}^{\mathrm{rec}}(v)\right\rangle = \int_0^\infty v\sigma_{\mathrm{f}}^{\mathrm{rec}}(v) f(v,T)dv \quad . \tag{5.4}$$

Como a recombinação poderá deixar o íon resultante em qualquer um de seus estados, teremos então que o coeficiente de recombinação total será uma soma sobre as contribuições individuais de todos os níveis:

$$\alpha_{\mathrm{Tot}}(T) = \sum_n \alpha_n(T) \quad . \tag{5.5}$$

Teremos então que os parâmetros relevantes a serem incluídos em nosso banco de dados atômicos para descrever os processos de fotoionização e recombinação direta serão as seções de choque de fotoionização a partir do estado fundamental e os coeficientes de recombinação totais. É importante frisar que estes dois dados não podem ser tomados de fontes independentes na literatura. O princípio do balanceamento detalhado (5.2) implica que ao aumentarmos a seção de choque de fotoionização em (5.1) estamos também aumentando a taxa de recombinação do outro lado da balança, fazendo com que as abundâncias fracionais do elemento em questão não sejam tão dependentes com as formas individuais das seções de choque. O mais importante é incluir coeficientes de recombinação consistentes com as seções de choque de fotoionização empregadas, de modo a não introduzir distorções no equilíbrio de ionização.

O processo de fotoionização na verdade pode se dar através de dois canais principais. No primeiro, o fóton ioniza o íon arrancando diretamente um elétron da camada de valência:

$$\mathrm{X}^{+i} + \gamma \rightarrow \mathrm{X}^{+i+1} + \mathrm{e}^- \quad , \tag{5.6}$$

sendo que o processo inverso é denominado *recombinação radiativa*.

A segunda possibilidade é um processo em duas etapas:

$$\mathrm{X}^{+i} + \gamma \rightarrow \mathrm{X}^{+i**} \rightarrow \mathrm{X}^{+i+1} + \mathrm{e}^{-} \quad , \tag{5.7}$$

ou seja, o íon é primeiro levado a um estado duplamente excitado auto-ionizante, em seguida decaindo espontaneamente e ejetando um dos elétrons. O processo inverso a esta reação denomina-se *recombinação dieletrônica*. Este processo, cuja importância foi ressaltada pela primeira vez por Burgess (1964), tipicamente é responsável por aumentar os coeficientes de recombinação a altas temperaturas por várias ordens de grandeza.

Por ser um processo ressonante, ele introduzirá inúmeros picos bastante localizados nas seções de choque de fotoionização. Como a maioria dos métodos empregados para calcular as seções de choque não delineiam estas estruturas de ressonâncias em detalhes, teremos que os coeficientes de recombinação derivados incluirão apenas a componente radiativa. A componente dieletrônica necessitará então ser calculada por algum outro método alternativo, e não será portanto auto-consistente com as seções de choque de fotoionização. Atualmente esta situação está mudando rapidamente com a introdução de métodos mais elaborados para incluir toda a estrutura de ressonâncias em detalhes, conforme veremos nas seções seguintes.

A recombinação dieletrônica é significativa apenas a temperaturas suficientemente altas para que a energia média dos elétrons $\frac{3}{2}kT$ seja comparável à energia das transições permitidas envolvendo o estado fundamental. Porém, entre o estado fundamental e o primeiro nível ligado a este por uma transição ressonante, podem existir outros níveis metaestáveis cujas transições proibidas também contribuirão para a recombinação dieletrônica. Este processo se manifestará em temperaturas mais baixas, e para alguns íons as taxas resultantes excederão as da componente radiativa (Storey 1981).

Levando-se em consideração todas as contribuições para o coeficiente de recombinação teremos então:

$$\alpha_{\mathrm{Tot}}(T) = \alpha_{\mathrm{rad}}(T) + \alpha_{\mathrm{baixa\ T}}(T) + \alpha_{\mathrm{alta\ T}}(T) \quad . \tag{5.8}$$

Na seção seguinte passamos a discutir as fontes disponíveis na literatura para cada uma dessas componentes dos coeficientes de recombinação, bem como para as seções de choque de fotoionização a partir do estado fundamental.

Ao longo de todo este capítulo ao falarmos do coeficiente de recombinação de um dado íon estamos, por convenção, nos referindo ao íon *recombinado* nas reações acima, X^{+i}.

5.2. Coleta de dados

O trabalho seminal para os coeficientes de recombinação foi feito por Aldrovandi & Péquignot (1973; 1976, daqui em diante AP). Nele os autores realizaram uma compilação de seções de choque de fotoionização, as quais utilizaram para derivar coeficientes de recombinação radiativa para os íons do He, C, N, O, Ne, Mg, Si e S. Os coeficientes de recombinação dieletrônica[☞] correspondentes foram calculados a partir da fórmula de Burgess (1965).

Woods, Shull, & Sarazin (1981) forneceram coeficientes de recombinação para os íons do Fe. Seus dados foram incluídos na compilação de Shull & van Steenberg (1982), que também estenderam o trabalho de AP para os íons do Ar, Ca e Ni. Os coeficientes de recombinação radiativa adotados foram os mesmos de AP, apenas interpolados para os íons do Ar, Ca e Ni. Porém, os coeficientes de recombinação dieletrônica foram todos revistos com base nos trabalhos de Jacobs et al. (1977; 1978; 1979; 1980), e foram ajustados por uma fórmula analítica prática e simples. Posteriormente Arnaud & Rothenflug (1985) revisaram os dados para alguns íons da seqüência isoeletrônica do He, Li e Be.

Landini & Monsignori Fossi (1990; 1991) empregaram a base de dados de Shull & van Steenberg e Arnaud & Rothenflug para interpolar os coeficientes de recombinação para os íons de outros elementos de menor abundância astrofísica: F, Na, Al, P, Cl, K, Ti, Cr, Mn e Co. Descobrimos que Landini & Monsignori Fossi cometeram um erro ao interpolar os coeficientes de recombinação radiativa de Arnaud & Rothenflug para os íons da seqüência isoeletrônica do Li e Be. Acreditamos que a origem deste erro deva-se ao fato de Arnaud & Rothenflug utilizarem uma convenção para se referir aos coeficientes de recombinação através do íon *recombinante* ao invés do íon *recombinado*, que é a maneira usual. Assim, os coeficientes de recombinação das seqüências isoeletrônicas do Li e Be foram erroneamente interpretadas por Landini & Monsignori Fossi como pertencentes as seqüências isoeletrônicas do He e Li respectivamente, como mostra a Figura 5.1.

Outras revisões que se sucederam foram os trabalhos de Péquignot, Petitjean, & Boisson (1991) com os coeficientes de recombinação radiativa para os íons do He, C, N, O e Ne, Arnaud & Raymond (1992) com os coeficientes de recombinação radiativa e dieletrônica para os íons do Fe e Mazzotta et al. (1998), que produziram uma nova revisão completa dos coeficientes de recombinação dieletrônica dos íons de todos os elementos do H ao Ni. Por fim, recentemente Mazzitelli & Mattioli (2002) extrapolaram a base de dados disponível para englobar também os elementos Cu, Zn, Ga e Ge.

[☞] Ao mencionarmos simplesmente "recombinação dieletrônica" estamos na verdade nos referindo apenas à componente de alta temperatura.

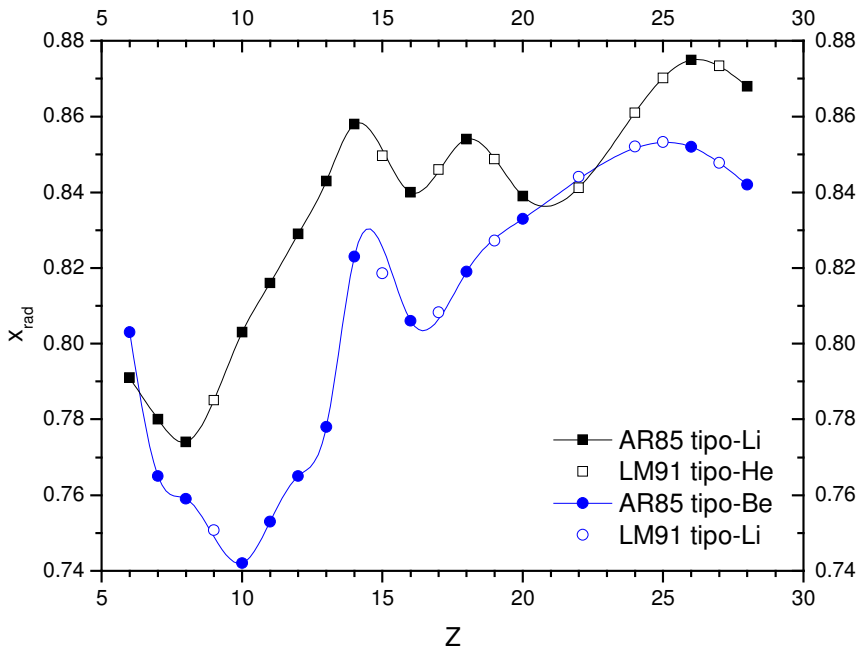


Figura 5.1: Parâmetros dos ajustes dos coeficientes de recombinação radiativa dos íons do tipo Li e Be.
Este gráfico mostra um dos parâmetros da fórmula de ajuste dos coeficientes de recombinação radiativa, $\alpha_{\rm rad} = A_{\rm rad}(T/10^4)^{x_{\rm rad}}$, como função da carga do núcleo do íon. Os símbolos cheios denotam os dados de Arnaud & Rothenflug (1985) para as seqüências isoeletrônicas do Li e Be, interligados por *splines* cúbicas. Os símbolos vazios correspondem aos dados de Landini & Monsignori Fossi (1990; 1991), que interpretaram erroneamente os dados de Arnaud & Rothenflug como sendo referente às seqüências isoeletrônicas do He e Li, respectivamente.

Podemos neste ponto notar que existe uma lacuna enorme na literatura se desejamos atingir o nosso objetivo de levantar os dados atômicos necessários para o cálculo do equilíbrio de ionização de todos os elementos do H ao Zn. Para a maior parte dos elementos de menor abundância astrofísica os únicos coeficientes de recombinação radiativa disponíveis foram derivados a partir de interpolações ou extrapolações do trabalho de AP. Embora estes dados possam ser úteis para aplicações em plasmas excitados colisionalmente, eles não se prestam à utilização em meios fotoionizados se não dispomos de seções de choque de fotoionização plenamente consistentes.

Uma compilação de seções de choque de fotoionização a partir do estado fundamental freqüentemente citada na literatura astronômica é a de Verner e colaboradores (Verner et al. 1993; Verner & Yakovlev 1995; Verner et al. 1996). Estes autores fornecem não apenas seções de choque a partir da camada de valência, mas também a partir da camadas internas, processo este que pode levar ainda à ejeção de elétrons Auger:

$$\mathrm{X}^{+i} + \gamma \rightarrow \mathrm{X}^{+i+1**} + \mathrm{e}^- \rightarrow \mathrm{X}^{+i+n} + n\mathrm{e}^- \quad , \tag{5.9}$$

com n podendo atingir até 8 a 10 em alguns íons complexos. Probabilidades de ejeção de elétrons Auger para os íons dos elementos do Be ao Zn estão listadas em Kaastra & Mewe (1993).

Infelizmente estes autores só se preocuparam em casar suas seções de choque de fotoionização com coeficientes de recombinação radiativa consistentes para as seqüências isoeletrônicas do H, He, Li e Na (Verner & Ferland 1996). Eles argumentam que para todos os íons os coeficientes de recombinação dieletrônica a baixa temperatura serão mais importantes do que os correspondentes radiativos, exceto para estas seqüências isoeletrônicas em particular, cujos íons não apresentam níveis metaestáveis de baixa energia.

De fato, a maior fonte de incerteza no cálculo do equilíbrio de ionização se origina da indisponibilidade de coeficientes de recombinação a baixas temperaturas, pois atualmente estes foram calculados apenas para os íons do C, N, O, Mg, Al, Si e Ne (Nussbaumer & Storey 1983, 1986, 1987). Porém, o argumento de Verner & Ferland não se justifica, pois não é verdade que a recombinação dieletrônica a baixas temperaturas sempre se sobressairá sobre a recombinação radiativa, além do que a recombinação radiativa sempre domina a temperaturas suficientemente baixas.

No código de fotoionização Cloudy (Ferland et al. 1998), amplamente empregado pela comunidade astronômica, estão embutidas as seções de choque de fotoionização e os coeficientes de recombinação radiativa de Verner e colaboradores. Para os íons não pertencentes às seqüências isoeletrônicas do H, He, Li e Na são empregados os coeficientes de recombinação das inúmeras compilações citadas acima, inconsistentes portanto com as seções de choque de fotoionização.

Outro conjunto importante de seções de choque de fotoionização foi gerado por uma colaboração internacional estabelecida na década de 80 conhecida como o Opacity Project (Seaton et al. 1994), com o objetivo de obter dados necessários para o cálculo de opacidades em envelopes estelares. Esta colaboração empregou o método sofisticado conhecido como método da matriz R (Burke, Hibbert, & Robb 1971; Burke & Robb 1975), o qual possibilita delinear toda a estrutura de ressonâncias das seções de choque em detalhes (obviamente a um custo computacional elevado se comparado a outros métodos mais simples). Ao todo, foram calculadas seções de choque de fotoionização a partir dos níveis com números quânticos $n \leq 10$ e $l \leq 4$ de todos os estágios de ionização dos elementos H-Si, S, Ar, Ca e Fe com até 24 elétrons. A grande massa de dados produzida se encontra disponível publicamente através do banco de dados TOPbase (Cunto & Mendoza 1992; Cunto et al. 1993).

Entretanto as seções de choque do OP não podem ser diretamente incorporadas aos códigos de fotoionização por dois motivos:

- A fim de delinear a estrutura de ressonâncias, as seções de choque são calculadas numa grade extensa de pontos de energia, às vezes envolvendo milhares de pontos. A inclusão de todos estes pontos traria um sério impacto no desempenho dos códigos de fotoionização,

onde as taxas de fotoionização são repetidamente calculadas em inúmeras iterações nas várias camadas da nuvem.

- As posições dos picos de ressonâncias estão mal calculadas, com incertezas nas energias da ordem de 1%. Embora possam parecer pequenos, os erros nas posições dos picos são muito maiores do que as suas larguras, o que poderia fazer com que fossem introduzidas coincidências espúrias com certas transições importantes. Se uma dessas ressonâncias for erroneamente deslocada para a freqüência correspondente à transição de Ly-α do H, por exemplo, a taxa de fotoionização estará incorreta por ordens de grandeza.

Verner e colaboradores valeram-se dos dados do OP para melhorar as suas seções de choque na região de energias próximas ao limiar de ionização (Verner, et al. 1996). As seções de choque de fotoionização do OP não são confiáveis na região de altas energias por empregar extrapolações simples em leis de potência, extremamente imprecisas. Para contornar as dificuldades acima, os autores adotaram o procedimento de ignorar completamente a estrutura de ressonâncias, se preocupando em ajustar somente o *background* das seções de choque.

Uma abordagem alternativa foi seguida por Bautista, Romano, & Pradhan (1998), que propuseram que as seções de choque do OP fossem suavizadas de modo a eliminar as incertezas dos picos de ressonâncias, porém retendo algumas estruturas localizadas que levam em conta a contribuição destas. O processo de suavização sugerido consiste em convoluir as seções de choque com uma Gaussiana de largura proporcional à freqüência:

$$\hat{\sigma}(\nu) = \frac{1}{\delta\sqrt{2\pi}} \int_{\nu_0}^{\infty} \sigma(\nu')\, e^{-\frac{(\nu-\nu')^2}{2\delta^2}}\, d\nu' \quad , \tag{5.10}$$

onde $\delta = \Delta \cdot \nu$, e Δ é uma constante. Na prática, Bautista et al. constataram que $\Delta = 0.03$ é uma escolha suficientemente conservadora face às incertezas nas posições das ressonâncias e que ao mesmo tempo resulta em seções de choque suaves, as quais podem ser aproximadas por um número reduzido de pontos para interpolação linear, como mostra a Figura 5.2.

Entretanto, gostaríamos de levantar aqui uma questão intrigante: as seções de choque não são naturalmente suavizadas no plasma astrofísico, uma vez que o meio deve enxergar uma seção de choque média devido ao movimento térmico do gás? Neste caso não seria mais aceitável fisicamente suavizar as seções de choque como função da temperatura?

Visando encontrar a resposta a essa pergunta, na seção seguinte passamos a calcular a forma da seção de choque suavizada percebida pelo meio.

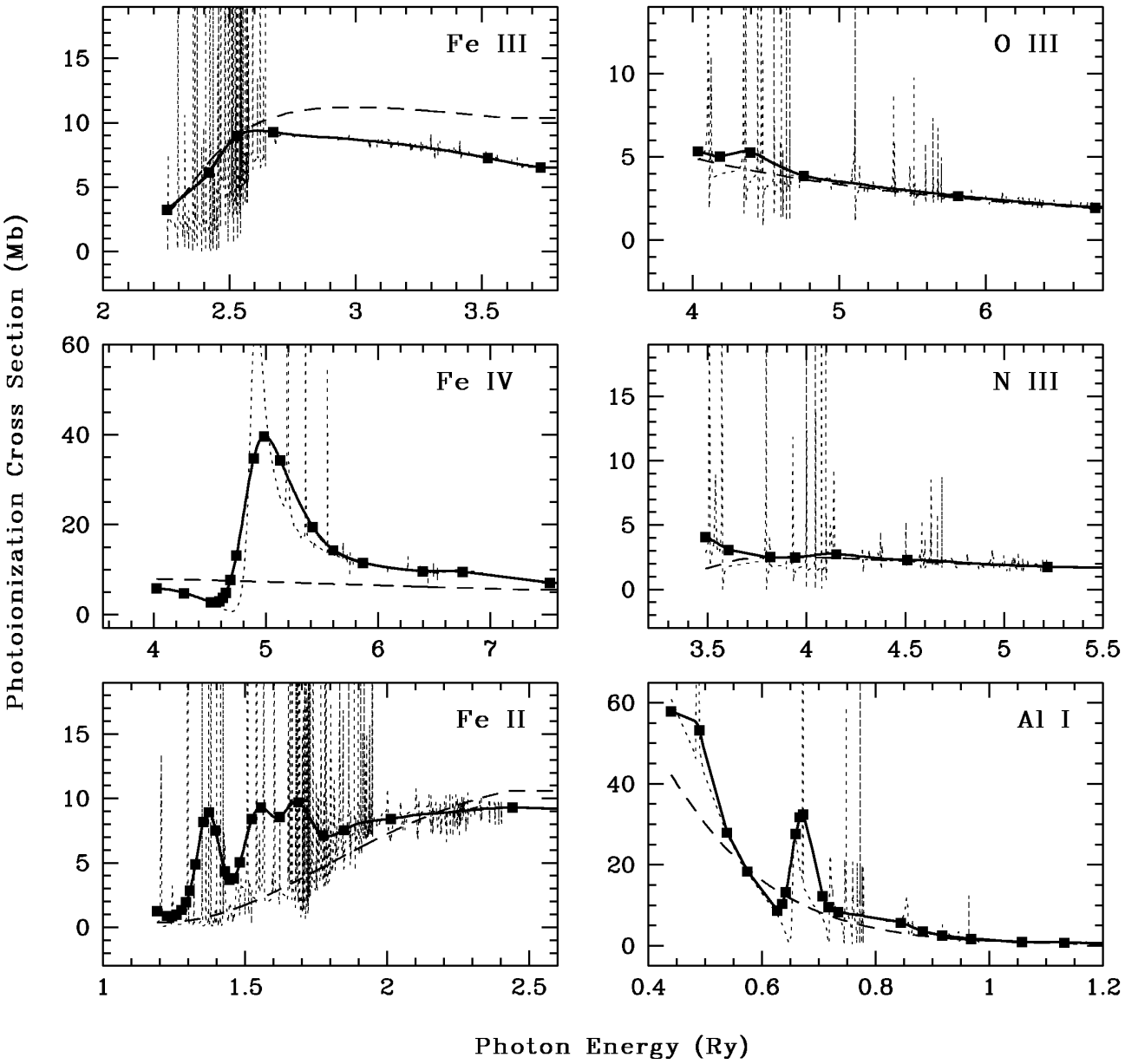


Figura 5.2: Seções de choque do OP suavizadas.
A largura da Gaussiana empregada na suavização corresponde a $\Delta = 0.03$. Para os íons do Fe os autores basearam-se em seções de choque melhores do que as do OP, discutidas mais adiante. As curvas tracejadas correspondem às seções de choque de Verner et al. (1996), as quais ignoram a estrutura de ressonâncias. Extraído de Bautista et al. (1998).

5.3. Suavização térmica

Para simplificar, vamos considerar inicialmente o caso em que o campo de radiação se propaga em uma única direção $\vec{x}$, e também que todas as partículas do meio se deslocam à mesma velocidade $\vec{v}$ constante. Os fótons terão então suas freqüências deslocadas de ν para ν' de acordo com:

$$\nu' = \nu \frac{c - v_x}{c} \quad , \tag{5.11}$$

onde v_x é a componente da velocidade na direção de propagação do campo.

Como estas serão as freqüências pelas quais o campo será percebido pelo meio, podemos então substituir as seções de choque de fotoionização pelas seções de choque modificadas $\sigma(\nu) \mapsto \hat{\sigma}(\nu)$:

$$\hat{\sigma}(\nu) = \sigma(\nu'(\nu)) \quad . \tag{5.12}$$

Para o caso geral no qual as partículas do meio são caracterizadas por uma distribuição de velocidades $f_x(v_x)$ teremos:

$$\hat{\sigma}(\nu) = \int_0^\infty \sigma(\nu') P_{\nu'}(\nu', \nu) d\nu' \quad , \tag{5.13}$$

onde $P_{\nu'}(\nu', \nu) d\nu'$ é a probabilidade de que um fóton emitido na freqüência ν seja visto pelo meio na freqüência entre ν' e $\nu' + d\nu'$, e é dada por:

$$P_{\nu'}(\nu', \nu) d\nu' = -f_x(v_x) dv_x \quad , \tag{5.14}$$

sendo que v_x, ν' e ν estão relacionados pela equação (5.11).

Se as partículas possuírem um movimento térmico aleatório a distribuição de velocidades será:

$$f_x(v_x) = \sqrt{\frac{m}{2\pi kT}}\, e^{-\frac{m v_x^2}{2kT}} \quad . \tag{5.15}$$

Juntando (5.15), (5.14) e (5.11):

$$P_{\nu'}(\nu', \nu) = \frac{c}{\nu} \sqrt{\frac{m}{2\pi kT}}\, e^{-\frac{mc^2}{2kT} \frac{(\nu' - \nu)^2}{\nu^2}} \quad . \tag{5.16}$$

Esta relação será válida para o caso em que o campo de radiação se propaga em todas as direções e também mesmo que ele seja anisotrópico, pois o movimento aleatório das partículas não assume nenhuma direção preferencial.

Substituindo (5.16) em (5.13):

$$\hat{\sigma}(\nu) = \frac{1}{\delta\sqrt{2\pi}} \int_0^\infty \sigma(\nu')\, e^{-\frac{(\nu' - \nu)^2}{2\delta^2}} d\nu' \quad , \tag{5.17}$$

onde $\delta = \Delta \cdot \nu$. Esta expressão é muito semelhante àquela proposta por Bautista et al., equação (5.10), sendo que agora o parâmetro Δ estará relacionado com a temperatura de acordo com:

$$\Delta = \sqrt{\frac{kT}{mc^2}} \quad . \tag{5.18}$$

Levando em conta que m = 0.511 MeV, e 1 eV = 1.16 10^4 K teremos $\Delta = 0.03$ apenas para temperaturas altíssimas da ordem de T = 5 10^6 K. Concluímos portanto que a suavização térmica das seções de choque de fotoionização será tipicamente um efeito pequeno se comparado às incertezas nas posições das ressonâncias nos cálculos do OP.

Um aspecto sutil surge ao aplicarmos a fórmula (5.17) na região próxima ao limiar de ionização, onde a seção de choque suavizada resultará em apenas metade do valor real, já que metade do perfil Gaussiano abrangerá freqüências para as quais a seção de choque real é nula. Fisicamente, isto é exatamente o que seria de se esperar, uma vez que metade das partículas estarão se afastando do campo, e portanto metade dos fótons que foram emitidos nas freqüências do limiar de ionização em realidade não serão absorvidos.

Entretanto este é um efeito indesejado se estamos apenas interessados em obter uma média da seção de choque nestes pontos a fim de suavizar as ressonâncias. Por este motivo em nossa aplicação é mais conveniente truncarmos a integral em (5.17) na freqüência limite:

$$\hat{\sigma}(\nu) = \frac{1}{f\,\delta\sqrt{2\pi}} \int_{\nu_0}^{\infty} \sigma(\nu')\, e^{-\frac{(\nu'-\nu)^2}{2\delta^2}}\, d\nu' \quad , \tag{5.19}$$

sendo que incluímos também um fator de normalização correspondente à área do perfil Gaussiano truncado em ν_0:

$$f = \frac{1}{2}[1 + \mathrm{erf}\,(\frac{\nu - \nu_0}{\delta\sqrt{2}})] \quad , \tag{5.20}$$

onde utilizamos a função erro:

$$\mathrm{erf}\,(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt \quad . \tag{5.21}$$

O fator f será ½ próximo ao limiar, aumentando para 1 à medida que nos afastamos dele.

5.4. Coeficientes de recombinação unificados

O processo de suavização exposto pode ser utilizado em seções de choque computadas através do método da matriz R para incorporá-las adequadamente nos códigos de fotoionização. Acontece que infelizmente não é possível obter coeficientes de recombinação consistentes com as seções de choque do OP devido ao fato de elas serem *totais*, *i.e.*, incluírem contribuições que deixam o íon fotoionizado não apenas no seu estado fundamental, mas também em qualquer um de seus estados excitados. De acordo com a aproximação nebular, praticamente todos os íons no plasma estarão povoados no estado fundamental, e é somente a partir deste que poderão ocorrer as reações de recombinação.

As seções de choque totais são apropriadas para calcular as taxas de fotoionização no equilíbrio de ionização, e por este motivo são elas que devem ser incluídas no banco de dados atômicos. Porém, a fim de aplicar o princípio do balanceamento detalhado para obter as seções de choque de recombinação, o cálculo das seções de choque de fotoionização precisa ser repetido para incluir apenas a contribuição do estado fundamental. Do contrário estaremos incluindo ressonâncias em excesso, as quais são o resultado de acoplamentos com os níveis excitados do íon residual, fazendo com que a componentes dieletrônicas dos coeficientes de recombinação sejam seriamente superestimadas.

Nos últimos anos um grupo de pesquisa na Universidade Estadual de Ohio tem se preocupado em obter de modo consistente coeficientes de recombinação a partir de cálculos sofisticados baseados no método da matriz R (Nahar & Pradhan 1994). Estes coeficientes automaticamente incluem, além da componente radiativa, também as componentes dieletrônicas, e por isso são os primeiros a serem plenamente consistentes com as seções de choque de fotoionização. Por negligenciar as ressonâncias, métodos mais simples permitiram derivar apenas a componente radiativa de modo consistente.

Além da questão importante acerca da auto-consistência, temos também que a divisão dos coeficientes de recombinação em contribuições separadas é artificial e não é motivada fisicamente, podendo levar a erros significativos em regiões onde nenhuma destas componentes se sobressai em relação às demais. A Figura 5.3 compara o coeficiente de recombinação "unificado" do O III com as outras componentes obtidas separadamente sob aproximações diferentes.

Como resultado do esforço de quase uma década, foram calculados coeficientes de recombinação para 45 íons abundantes astrofisicamente: todos os íons do C, N (Nahar & Pradhan 1997; Nahar, Pradhan, & Zhang 2000) e O (Nahar 1999); Si I, Si II, S II, S III, F IV, Ne V, Na VI, Mg VII, Al VIII, Si IX e S XI (Nahar 1995, 1996a); Ar V, Ca VII e Fe XIII (Nahar 2000); Fe I (Nahar, Bautista, & Pradhan 1997), Fe II (Nahar 1997), Fe III (Nahar 1996b), Fe IV (Nahar, Bautista, & Pradhan 1998), Fe V (Nahar & Bautista 1999), Fe XVII (Pradhan, Nahar, & Zhang

2001; Zhang, Nahar, & Pradhan 2001), Fe XXIV-XXVI (Nahar, Pradhan, & Zhang 2001a) e Ni II (Nahar & Bautista 2001).

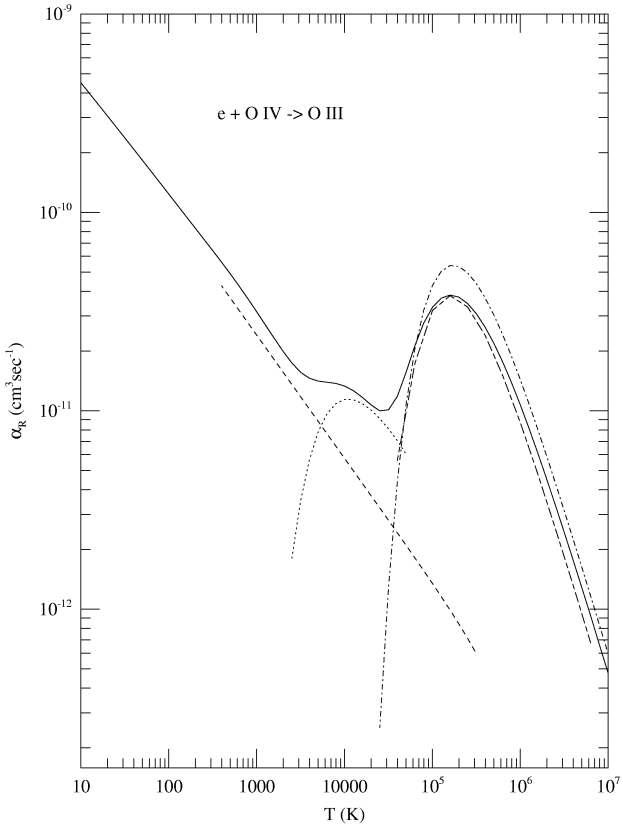


Figura 5.3: Coeficiente de recombinação unificado para o O III.
A linha contínua mostra o coeficiente de recombinação unificado (Nahar 1999), a linha tracejada a componente radiativa (Péquignot, et al. 1991), a linha pontilhada a componente dieletrônica a baixas temperaturas (Nussbaumer & Storey 1983), e as linhas com ponto-traço e traço longo-curto a componente dieletrônica a altas temperaturas (Shull & van Steenberg 1982; Badnell & Pindzola 1989, respectivamente). Adaptado de Nahar (2001).

Para que pudéssemos incorporar os coeficientes de recombinação unificados de maneira consistente em nosso banco de dados atômicos, estabelecemos contato com a pesquisadora Sultana Nahar, a qual nos forneceu as seções de choque totais de fotoionização para todos os íons mencionados acima.

Para pôr em prática o método de suavização descrito na seção 5.3 tivemos de desenvolver nossa própria rotina, pois aquelas encontradas em bibliotecas numéricas em geral utilizam algoritmos de integração válidos apenas para funções que variam lentamente. Em nossa aplicação, de um ponto a outro a seção de choque pode variar por ordens de grandeza ao passar por uma ressonância. A nossa rotina em Fortran adiciona cuidadosamente a contribuição de todos os intervalos, tomando o cuidado de resolver o perfil Gaussiano em pontos intermediários nos quais a curva é interpolada linearmente.

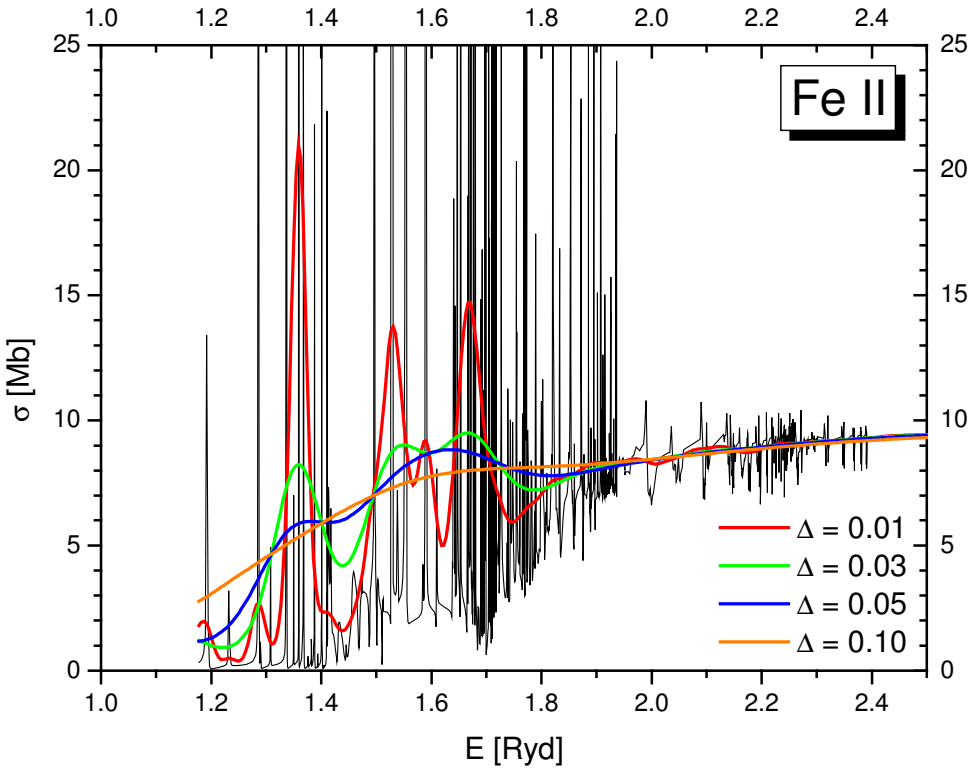


Figura 5.4: Seções de choque suavizadas para o Fe II.
Podemos notar que quanto menor o parâmetro de alargamento do perfil Gaussiano, mais proximamente a curva resultante seguirá a estrutura de ressonâncias da curva original. Para larguras muito grandes todas as estruturas localizadas desaparecem por completo. Confronte a curva com $\Delta = 0.03$ com a da Figura 5.2 .

A Figura 5.4 mostra resultados-teste para as seções de choque suavizadas do Fe II para vários parâmetros de alargamento. Comparando a curva com $\Delta = 0.03$ com a que aparece na Figura 5.2 podemos constatar o excelente acordo de nossos resultados com os obtidos por Bautista et al. (1998), o qual também utilizou os dados do grupo de Ohio para os íons Fe I-V. Concordamos com estes autores na escolha do parâmetro de alargamento, e passamos a adotar $\Delta = 0.03$ daqui em diante.

Para a grande maioria dos íons estudados, as ressonâncias não desempenham nenhum papel importante nas seções de choque suavizadas, as quais apenas acompanham o *background* das seções de choque originais. O *background* das seções de choque calculadas pelo grupo de Ohio é muito semelhante ao das seções de choque do OP, pelo que as nossas curvas suavizadas concordam bastante bem com as obtidas por Bautista et al. A principal diferença entre os dois conjuntos de dados é que os cálculos do grupo de Ohio são mais elaborados, o que se traduz numa estrutura de ressonâncias mais detalhada. A Figura 5.5 ilustra bem os resultados das comparações entre as diversas seções de choque. Gráficos semelhantes para todos os íons analisados constam no CD em anexo que acompanha a tese.

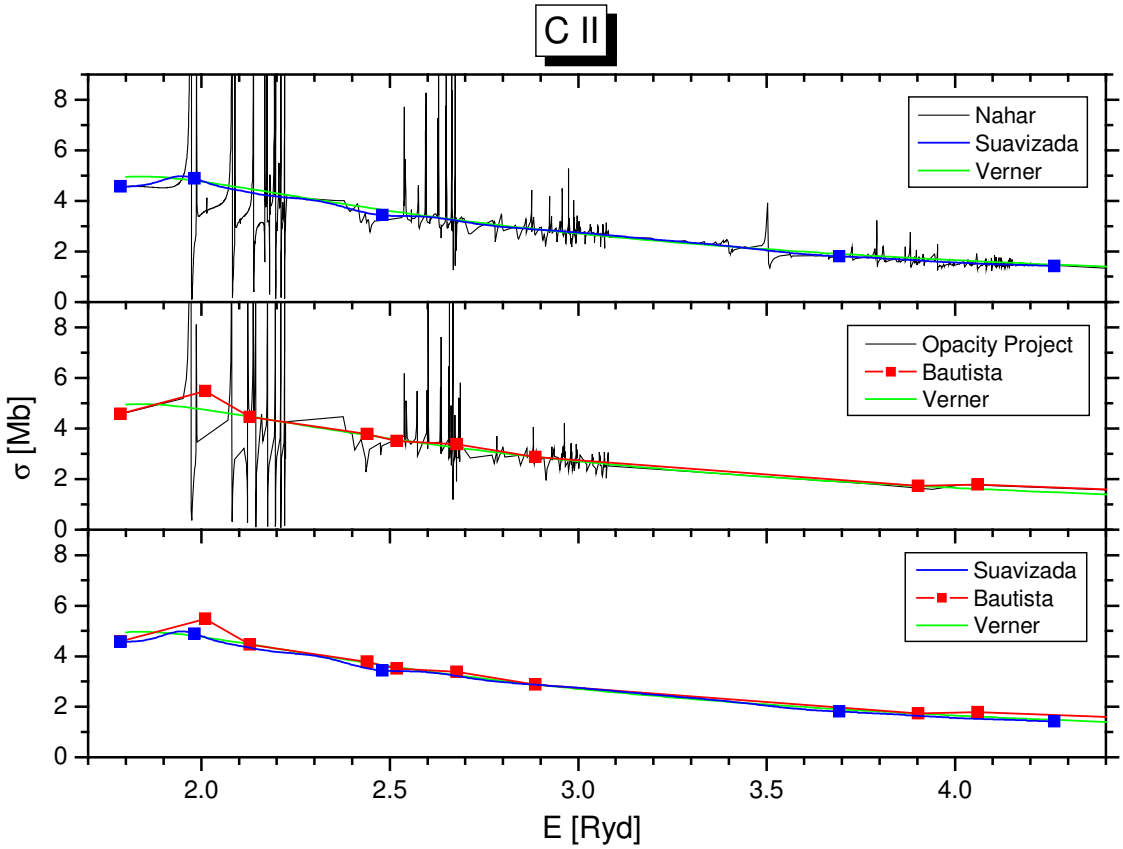


Figura 5.5: Seções de choque suavizadas para o C II.
Neste gráfico comparamos as nossas seções de choque suavizadas, baseadas nos dados de Nahar e colaboradores, com as obtidas por Bautista et al. (1998) com base nos dados do OP. Os quadrados descrevem as curvas suavizadas para interpolação linear. A curva azul é a nossa seção de choque suavizada, ao passo que a vermelha é apenas a interpolação linear dos pontos de Bautista et al. Também incluímos na comparação os ajustes analíticos de Verner et al. (1996).

Uma vez que as ressonâncias não são relevantes para as seções de choque que entram no cálculo das taxas de fotoionização, acreditamos que para a componente dieletrônica dos coeficientes de recombinação o mais importante seja precisão individual ao invés da auto-consistência. A única exceção seriam certos íons complexos com muitos elétrons, como o Fe I e Fe II, onde a estrutura de ressonâncias aparentemente é indistinguível do *background* (*cf.* Figura 5.4). Também estes seriam os únicos casos em que as seções de choque de Verner et al. (1996) não conduziriam a resultados satisfatórios.

Talvez a questão da consistência da componente dieletrônica dos coeficientes de recombinação venha a se tornar importante novamente quando os picos das ressonâncias tiverem suas posições bem estabelecidas e as seções de choque puderem ser diretamente incorporadas nos códigos de fotoionização.

Ao longo de nosso trabalho em suavizar as seções de choque do grupo de Ohio, nos deparamos com dificuldades que colocam em questão a confiabilidade dos resultados obtidos com os cálculos da matriz R atuais. O problema que trazemos à tona diz respeito à resolução dos picos de ressonâncias.

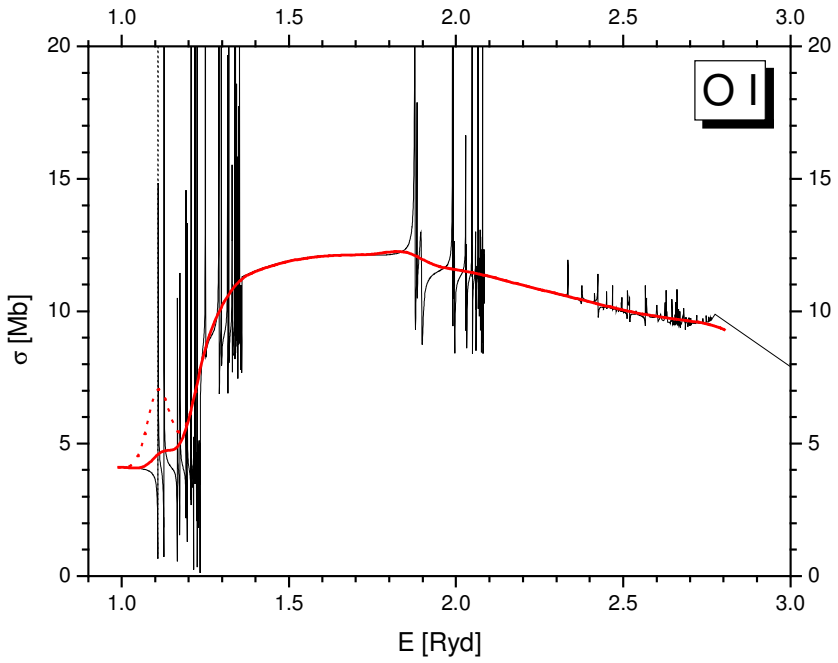


Figura 5.6: Seção de choque suavizada para o O I.
A saliência presente na seção de choque suavizada (linha pontilhada) é devido a uma ressonância mal-resolvida, e desaparece totalmente ao excluirmos um único ponto da seção de choque original.

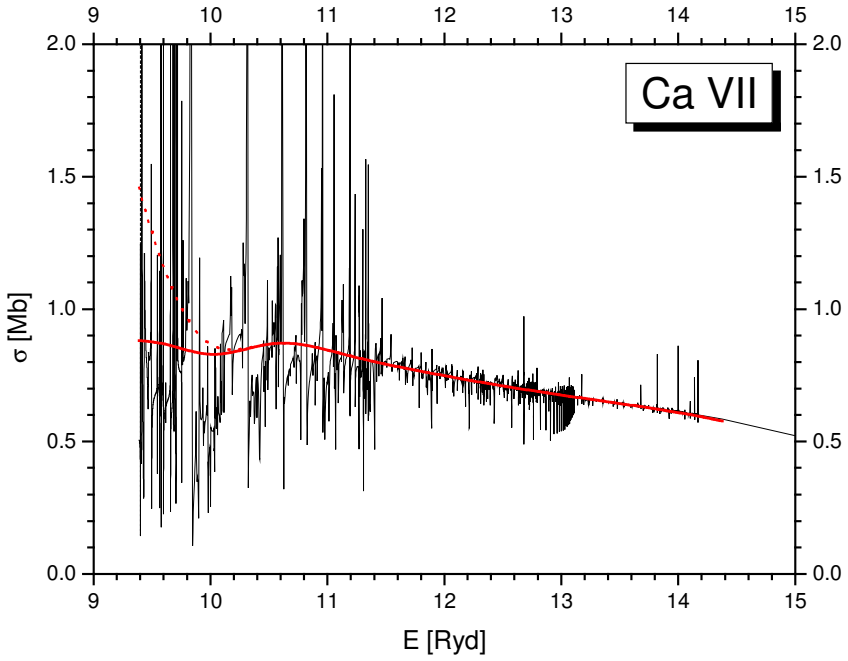


Figura 5.7: Seção de choque suavizada para o Ca VII.
A saliência presente na seção de choque suavizada (linha pontilhada) é devido a uma ressonância mal-resolvida, e desaparece totalmente ao excluirmos um único ponto da seção de choque original.

Nas figuras 5.6 e 5.7 podemos notar uma saliência nas seções de choque suavizadas do O I e Ca VII nas regiões próximas ao limiar de ionização. Elas são causadas por picos de ressonâncias mal-resolvidos, uma vez que desaparecem por completo ao excluirmos apenas um ponto nas seções de choque originais. Estas saliências ocupam uma fração considerável da área total da seção de

choque e possivelmente são espúrias, uma vez que estruturas similares não são evidentes nas seções de choque suavizadas de outros íons simples.

Em particular, os íons do tipo Li são casos gravíssimos, como podemos notar nas figuras 5.8 e 5.9 . O salto nas seções de choque em altas energias corresponde ao limiar de ionização da camada interna 1s, sobre o qual converge uma estrutura bastante proeminente de ressonâncias. No C IV a seção de choque suavizada exibe uma protuberância enorme devido a duas ressonâncias (em detalhes, no quadro interior). Note também a pequena saliência discretamente situada entre a protuberância e o salto na seção de choque.

No gráfico do O VI esta protuberância também está presente, porém a ressonância associada é constituída por apenas um único ponto! A pequena saliência visível no C IV não se manifesta aqui. A suavização da seção de choque do N V revela algo inesperado: a protuberância encolhe para uma saliência modesta, e aquela outra saliência presente no C IV agora dá lugar a uma estrutura proeminente gerada por uma ressonância mal-resolvida composta de um ponto apenas. Como o N V está colocado entre o C IV e o O VI na mesma seqüência isoeletrônica, acreditamos que a protuberância apenas não surgiu porque a grade de pontos de energia é grosseira demais e não coincidiu com a posição das ressonâncias.

O Fe XXIV é o caso mais drástico: as protuberâncias são tão grandes a ponto de toda a estrutura do *background* da seção de choque ficar encolhida quase que fora de escala no gráfico. A protuberância maior pode ser reduzida excluindo-se apenas um ponto na seção de choque, e ainda eliminada totalmente excluindo-se dois outros pontos (em detalhes, no quadro interior).

As ressonâncias dos íons do tipo Li foram posteriormente reconsideradas pelo grupo de Ohio (Pradhan 2000; Nahar, Pradhan, & Zhang 2001b) para as seções de choque do estado fundamental. Evidentemente todas as seções de choque dos níveis excitados precisam ser analisadas para que os coeficientes de recombinação sejam revistos.

As seções de choque de fotoionização do OP também não estão imunes ao problema de resolução das ressonâncias. Na verdade, acreditamos que este venha a se manifestar de maneira até mais acentuada, uma vez que as seções de choque do OP estão computadas numa grade de pontos 10 vezes mais grosseira do que nos dados do grupo de Ohio. Retornando à Figura 5.2, podemos reparar que a saliência presente na seção de choque do Al I também aparenta ser o resultado espúrio de uma ressonância mal-resolvida. Pudemos confirmar esta hipótese ao extrair esta seção de choque do banco de dados TOPbase e aplicar o processo de suavização, como mostra a Figura 5.10 .

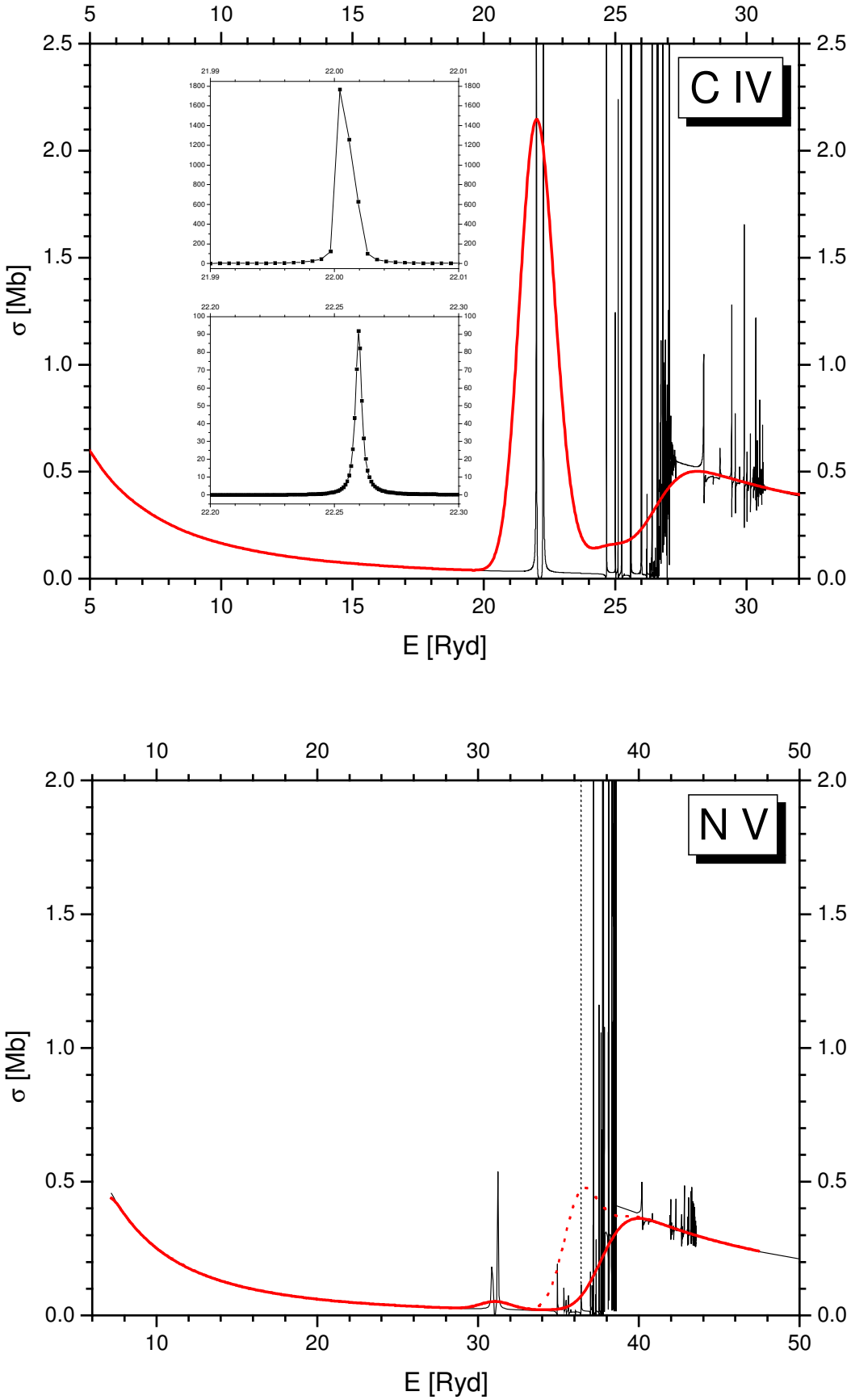


Figura 5.8: Seções de choque suavizadas para o C IV e N V.

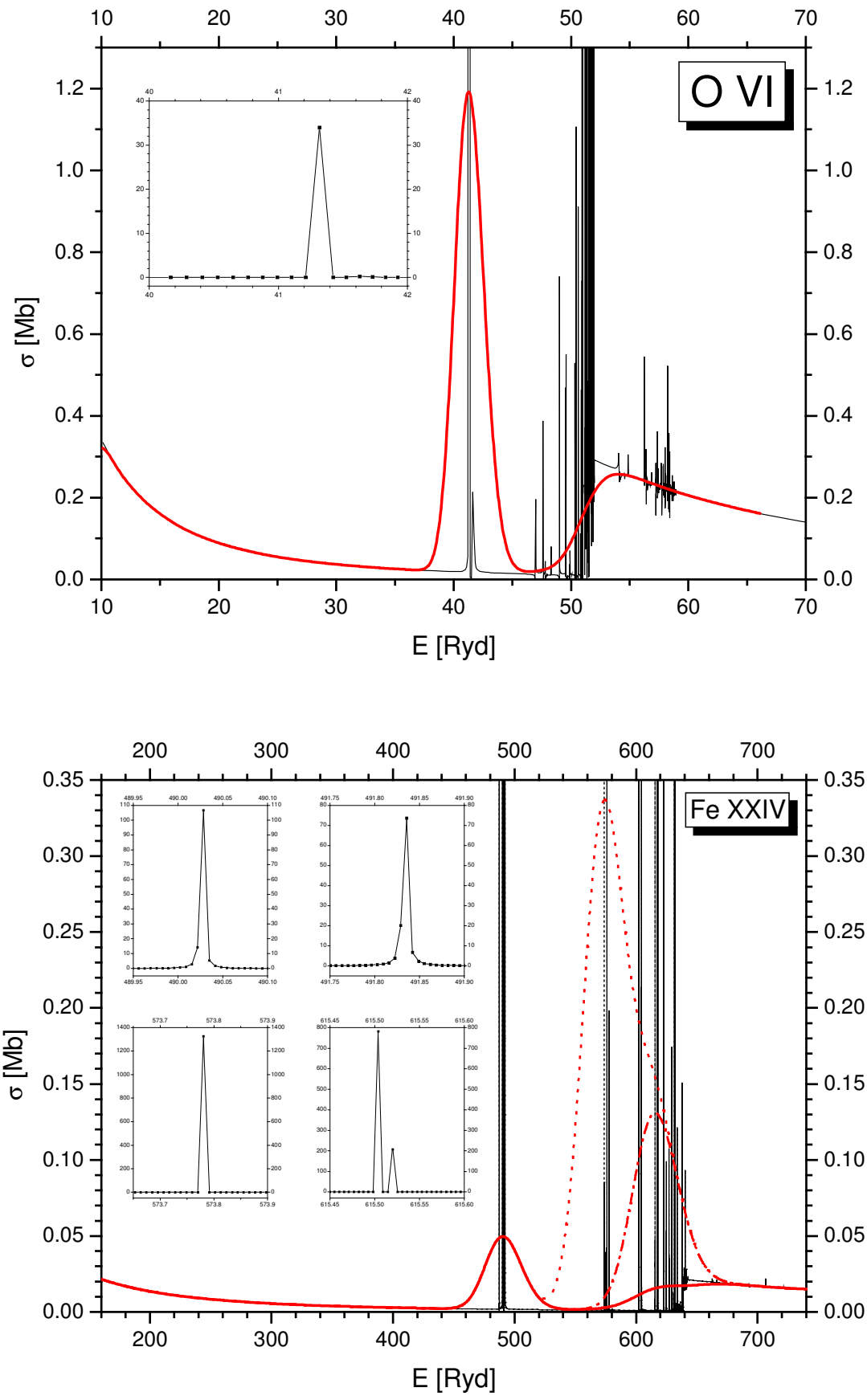


Figura 5.9: Seções de choque suavizadas para o O VI e Fe XXIV.

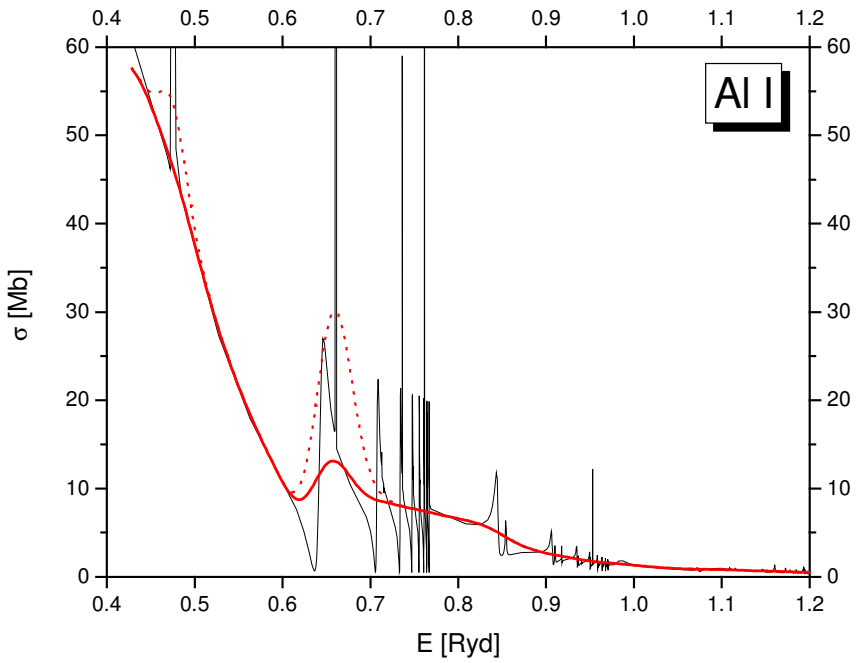


Figura 5.10: Seção de choque suavizada para o Al I.
As saliências presentes na seção de choque suavizada (linha pontilhada) se devem a ressonâncias mal-resolvidas, e desaparecem completamente ao excluirmos um único ponto da seção de choque original, a qual foi extraída do banco de dados TOPbase do OP.

Contestamos portanto o ponto de vista expresso em Bautista et al. (1998) segundo o qual a estrutura de ressonâncias contribui significativamente na seções de choque, e reiteramos a nossa conclusão de que para a grande maioria dos íons simples é suficiente incluir apenas a contribuição do *background*. Nos casos em que a suavização das seções de choque resultou em estruturas localizadas proeminentes, estas foram apenas devido a ressonâncias mal-resolvidas.

Nós não questionamos a realidade destas estruturas localizadas para alguns íons específicos, tais como os da seqüência isoeletrônica do Li discutidos acima e outros íons complexos como o Fe I e Fe II. Muito menos duvidamos da importância em incluir nas seções de choque a contribuição dessas estruturas. Argumentamos porém, que as seções de choque suavizadas envolverão incertezas muito grandes decorrentes da má resolução das ressonâncias. Essa dificuldade impede que sejamos capazes de prever de maneira robusta a forma geral das seções de choque suavizadas, uma vez que ela é alterada drasticamente com a exclusão de alguns poucos pontos nos dados originais, como mostra a Figura 5.11 para o Fe II.

O procedimento de suavização revelou-se uma ferramenta poderosa para apanhar ressonâncias não resolvidas, ao fazer com que as suas áreas sejam redistribuídas sobre uma região mais ampla. Este é, em realidade, o único efeito produzido pela convolução com o perfil Gaussiano normalizado. Teríamos então que mesmo que fosse possível incluir as seções de choque originais nos códigos de plasma, obteríamos um erro muito grande ao derivar as taxas de fotoionização, uma vez que uma fração significativa da área total da seção de choque estaria concentrada em picos de ressonância mal-resolvidos.

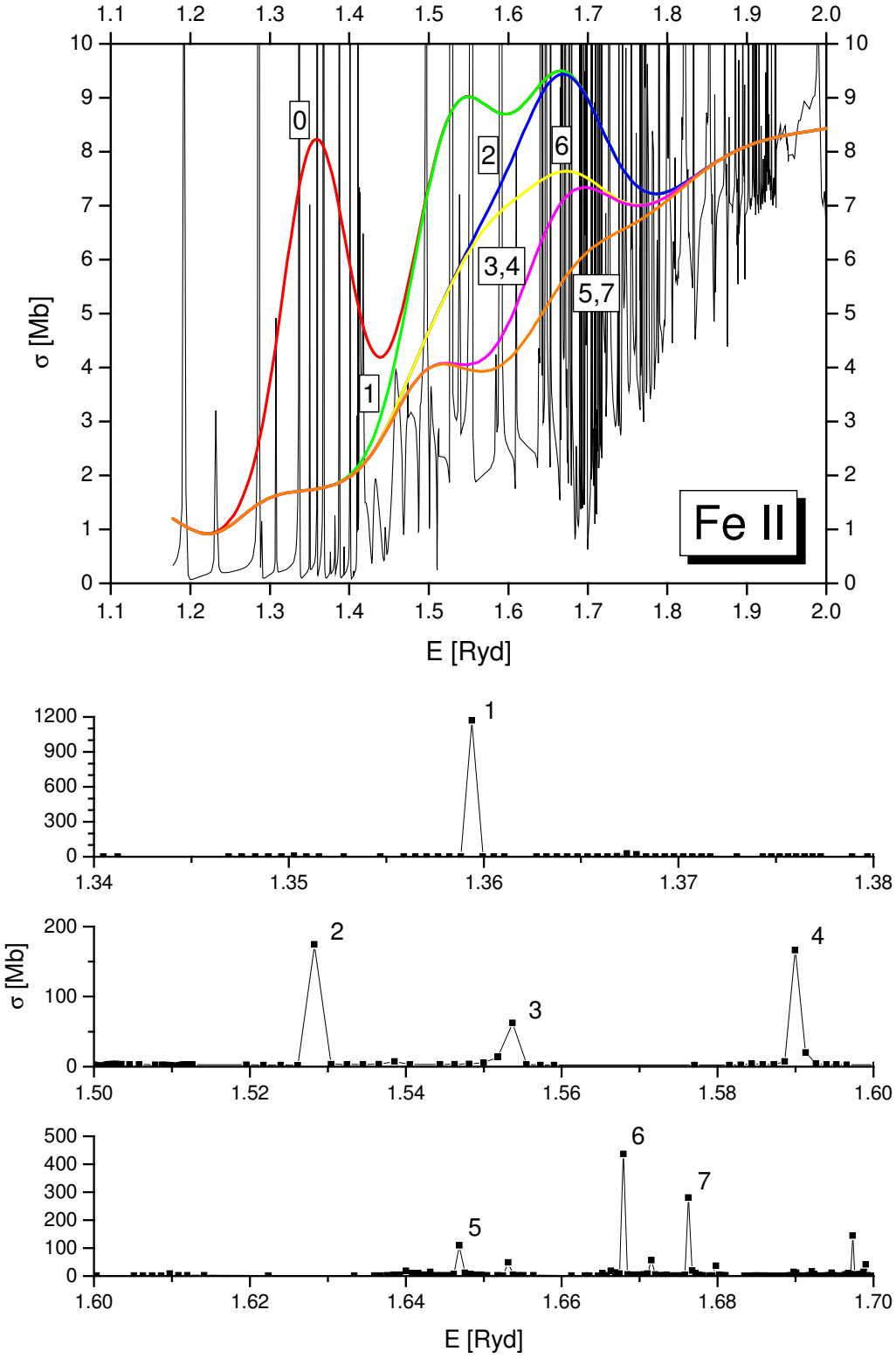


Figura 5.11: Picos de ressonância mal-resolvidos na seção de choque do Fe II.

As estruturas localizadas nas seções de choque suavizadas do Fe II (figuras 1.2 e 1.4) são geradas por inúmeros picos de ressonâncias mal-resolvidos, como podemos ver no painel de baixo. No painel de cima, mostramos o efeito produzido na seção de choque suavizada ao excluírmos gradativamente cada um dos pontos numerados no painel de baixo.

Não há motivos para supor que a convolução das seções de choque com uma distribuição Maxwelliana de velocidades também não introduza erros nos coeficientes de recombinação. Por isso acreditamos que há uma necessidade imperativa de aprimorar os cálculos com o método da matriz R atuais a fim de lidar com a questão da resolução das ressonâncias com mais cuidado.

Recentemente Ramírez & Bautista (2002) desenvolveram um complemento ao método da matriz R que ataca diretamente esse problema. Estes autores sugeriram uma maneira de estimar previamente as posições e larguras dos picos de ressonância, possibilitando gerar uma grade de pontos de energia mais otimizada a fim de resolvê-los adequadamente. Desto modo, um conjunto reduzido de pontos de energia bem escolhidos conduziria a resultados melhores do que um grande número de pontos sem muita correlação com a estrutura de ressonâncias. A nosso ver, o poder computacional disponível poderia então ser investido em sofisticar outros aspectos dos cálculos do método da matriz R, como a completeza da expansão de auto-estados nas funções de onda, por exemplo.

As deficiências que apontamos com relação à resolução dos picos de ressonâncias nas seções de choque atuais deverão servir como um estímulo à adoção urgente de métodos como o proposto por Ramírez e Bautista (2002).

5.5. Conclusões

Infelizmente não foi possível atingir o nosso objetivo proposto inicialmente de reunir um conjunto completo e auto-consistente de seções de choque de fotoionização e coeficientes de recombinação para todos os íons dos elementos do H ao Zn.

Como vimos na seção 5.2, a base de dados disponível contém inúmeras lacunas que precisam ser preenchidas a fim de possibilitar o cálculo do equilíbrio de ionização dos elementos de maneira consistente em plasmas fotoionizados. Para a grande maioria dos elementos do H ao Zn os coeficientes de recombinação foram apenas interpolados/extrapolados a partir dos dados de elementos mais abundantes. Para estes é necessário derivar novos coeficientes de recombinação radiativos consistentes com as seções de choque de fotoionização disponíveis na literatura, como as calculadas por Verner et al. (1996).

Também encontramos dificuldades ao tentar incorporar, para os elementos mais abundantes, o conjunto de dados auto-consistentes produzido pelo grupo de Ohio. A resolução das estruturas de ressonâncias é uma questão delicada cuja importância parece ter sido subestimada nos cálculos atuais do método da matriz R. É preciso investigar até que ponto os coeficientes de recombinação são afetados por este problema antes que possamos incorporar esse conjunto de dados nos códigos de fotoionização de maneira confiável.

Possivelmente uma nova geração de dados produzidos com o método da matriz R poderá estar disponível num futuro próximo, a qual incorporaria as melhorias sugeridas por Ramírez e

Bautista (2002). Outra motivação deverá vir de dados experimentais de seções de choque de fotoionização obtidos em fontes de luz síncrotron, ou ainda de seções de choque de recombinação medidas em anéis de armazenamento de íons, as quais estão atingindo a precisão necessária para calibrar os cálculos teóricos. A literatura neste tópico é extensa, pelo que remetemos o leitor a artigos de revisão recentes na literatura (Schippers et al. 2001; West 2001; Schippers 2003).

Referências

Aldrovandi, S. M. V., & Péquignot, D. 1973, A&A, 25, 137

---. 1976, A&A, 47, 321 (errata)

Arnaud, M., & Raymond, J. 1992, ApJ, 398, 394

Arnaud, M., & Rothenflug, R. 1985, A&AS, 60, 425

Badnell, N. R., & Pindzola, M. S. 1989, Phys. Rev. A, 39, 1690

Bautista, M. A., Romano, P., & Pradhan, A. K. 1998, ApJS, 118, 259

Burgess, A. 1964, ApJ, 139, 776

---. 1965, ApJ, 141, 1588

Burke, P. G., Hibbert, A., & Robb, W. D. 1971, J. Phys. B, 4, 153

Burke, P. G., & Robb, W. D. 1975, Adv. Atom. Mol. Phys., 11, 143

Cunto, W., & Mendoza, C. 1992, Rev. Mexicana Astron. Astrof., 23, 107

Cunto, W., Mendoza, C., Ochsenbein, F., & Zeippen, C. J. 1993, A&A, 275, L5

Ferland, G. J., Korista, K. T., Verner, D. A., Ferguson, J. W., Kingdon, J. B., & Verner, E. M. 1998, PASP, 110, 761

Jacobs, V. L., Davis, J., Kepple, P. C., & Blaha, M. 1977, ApJ, 215, 690

Jacobs, V. L., Davis, J., Rogerson, J. E., & Blaha, M. 1978, J. Quant. Spectrosc. Radiat. Transfer, 19, 591

---. 1979, ApJ, 230, 627

Jacobs, V. L., Davis, J., Rogerson, J. E., Blaha, M., Cain, J., & Davis, M. 1980, ApJ, 239, 1119

Kaastra, J. S., & Mewe, R. 1993, A&AS, 97, 443

Landini, M., & Monsignori Fossi, B. C. 1990, A&AS, 82, 229

---. 1991, A&AS, 91, 183

Mazzitelli, G., & Mattioli, M. 2002, ADNDT, 82, 313

Mazzotta, P., Mazzitelli, G., Colafrancesco, S., & Vittorio, N. 1998, A&AS, 133, 403

Nahar, S. N. 1995, ApJS, 101, 423

---. 1996a, ApJS, 106, 213 (errata)

---. 1996b, Phys. Rev. A, 53, 2417

---. 1997, Phys. Rev. A, 55, 1980

---. 1999, ApJS, 120, 131

---. 2000, ApJS, 126, 537

---. 2001, in *"Planetary nebulae: their evolution and role in the universe"*, eds. S. Kwok, M. Dopita, & R. Sutherland (Canberra, Austrália: Proc. IAU Symposium, vol. 209)(arXiv astro-ph/0202343)

Nahar, S. N., & Bautista, M. A. 1999, ApJS, 120, 327

---. 2001, ApJS, 137, 201

Nahar, S. N., Bautista, M. A., & Pradhan, A. K. 1997, ApJ, 479, 497

---. 1998, Phys. Rev. A, 58, 4593

Nahar, S. N., & Pradhan, A. K. 1994, Phys. Rev. A, 49, 1816

---. 1997, ApJS, 111, 339

Nahar, S. N., Pradhan, A. K., & Zhang, H. L. 2000, ApJS, 131, 375

---. 2001a, ApJS, 133, 255

---. 2001b, Phys. Rev. A, 63, 060701

Nussbaumer, H., & Storey, P. J. 1983, A&A, 126, 75

---. 1986, A&AS, 64, 545

---. 1987, A&AS, 69, 123

Osterbrock, D. E. 1989, *"Astrophysics of Gaseous Nebulae and Active Galactic Nuclei"* (Mill Valley, CA: University Science Books)

Péquignot, D., Petitjean, P., & Boisson, C. 1991, A&A, 251, 680

Pradhan, A. K. 2000, ApJ, 545, L165

Pradhan, A. K., Nahar, S. N., & Zhang, H. L. 2001, ApJ, 549, L265

Ramírez, J. M., & Bautista, M. A. 2002, J. Phys. B, 35, 4139

Schippers, S. 2003, (arXiv astro-ph/0308544)

Schippers, S., et al. 2001, in *"Stellar coronae in the Chandra and XMM-Newton era"*, eds. F. Favata, & J. J. Drake (Noordwijk, Holanda: ASP Conference Series, vol. 277), p. 591 (arXiv astro-ph/0301170)

Seaton, M. J., Yan, Y., Mihalas, D., & Pradhan, A. K. 1994, MNRAS, 266, 805

Shull, J. M., & van Steenberg, M. 1982, ApJS, 48, 95 (veja errata em ApJS 49, 351)

Storey, P. J. 1981, MNRAS, 195, 27p

Verner, D. A., & Ferland, G. J. 1996, ApJS, 103, 467

Verner, D. A., Ferland, G. J., Korista, K. T., & Yakovlev, D. G. 1996, ApJ, 465, 487

Verner, D. A., & Yakovlev, D. G. 1995, A&AS, 109, 125

Verner, D. A., Yakovlev, D. G., Band, I. M., & Trzhaskovskaya, M. B. 1993, ADNDT, 55, 233

West, J. B. 2001, J. Phys. B, 34, R45

Woods, D. T., Shull, J. M., & Sarazin, C. L. 1981, ApJ, 249, 399 (veja errata em ApJS 49, 351)

Zhang, H. L., Nahar, S. N., & Pradhan, A. K. 2001, Phys. Rev. A, 64, 032719

6. Programação Orientada a Objetos

6.1. Complexidade

Códigos de fotoionização são amplamente utilizados pelos astrônomos para modelar as suas observações. Uma vez que tais códigos realizam uma simulação numérica extremamente complexa, que garantias temos de que o resultado final é confiável e não está comprometido por *bugs*? Vejamos o que diz Ferland, acerca do código de fotoionização Cloudy, amplamente empregado pela comunidade astronômica (Ferland 1996):

"Eu acredito que a questão da confiabilidade ante à complexidade será gradativamente o maior fator limitante ao desenvolvimento de simulações numéricas em grande escala. Novos métodos de escrever código terão que ser desenvolvidos a fim de tirar o máximo proveito do poder computacional de máquinas futuras. As máquinas estão se tornando mais poderosas muito mais rapidamente do que as pessoas estão se tornando mais espertas."

Certamente os astrônomos não foram os únicos e possivelmente nem os primeiros a constatarem este problema. Qualquer cientista que realize simulações numéricas em grande escala pôde constatar o aumento dramático ocorrido no poder computacional disponível. Há décadas os computadores vêm se tornando mais rápidos num crescimento vertiginoso e exponencial ditado pela lei de Moore (Moore 1965), segundo a qual o poder computacional disponível para uma quantidade fixa de dólares dobra a cada 18 meses.

E com mais poder computacional à disposição abrem-se novas possibilidades em atacar problemas maiores, e por conseguinte, também mais complexos. Por exemplo, podemos pensar numa simulação de N-corpos, na qual aumentamos o tamanho da simulação de 100 para 1000 partículas. Entretanto, tipicamente não desejamos apenas aumentar o tamanho da simulação, mas também a tornaremos mais sofisticada incluindo um campo magnético, por exemplo. Assim, concluímos que problemas maiores em geral implicam no aumento da complexidade.

E o que temos feito para lidar com essa complexidade? Atualmente esta questão atingiu proporções tão sérias que se aventurar pelos códigos científicos mais complexos é algo semelhante a nadar num mar revolto povoado de criaturas marinhas, tentando desesperadamente evitar que uma delas nos agarre e nos arraste direto para o fundo.

Essa situação decorre do fato de que há anos os cientistas vêm empregando ferramentas que não são adequadas para lidar com problemas muito grandes, pois em geral resultam em códigos que são:

- difíceis de manter;
- difíceis de reutilizar em problemas semelhantes, ou ainda estender a novos problemas,
- difíceis de entender e dividir com colaboradores.

Será que estas são as conseqüências naturais decorrentes do aumento da complexidade, as quais infelizmente teremos que aceitar e aprender a conviver? Certamente não. Um código de fotoionização típico contém cerca de 10^4-10^5 linhas de código; já um conjunto de aplicativos de escritório como o Star Office contém nada menos do que 7,5 10^6 de linhas de código! Códigos desta escala não são obra de apenas um desenvolvedor isoladamente, mas sim são o resultado de um trabalho em equipe de vários programadores-mês.

Seguramente os cientistas de computação já se depararam com os mesmos problemas dos astrônomos de hoje há décadas atrás, e desde então desenvolveram as ferramentas adequadas para lidar com a complexidade. O que devemos fazer então, ao contrário do que diz a afirmação bombástica acima, não é desenvolver novos métodos de escrever código, e sim apenas aprender a utilizar o ferramental criado pelos cientistas de computação, mais espertos do que nós astrônomos.

As ferramentas que procuramos explorar no decorrer deste trabalho foram a *programação orientada a objetos* e a *UML*. A programação orientada a objetos (POO) consiste numa abordagem de programação que resulta em códigos menos suscetíveis às dificuldades enumeradas acima, ou seja, mais fáceis de manter, reutilizar, estender, compreender e dividir com colaboradores. A UML (acrônimo em inglês de *Unified Modeling Language*) é uma linguagem gráfica que se tornou o padrão adotado no meio acadêmico e na indústria para a modelagem e especificação de sistemas orientados a objetos. Através da UML podemos criar um "organograma" do nosso projeto, o qual desempenhará o mesmo papel que a planta de um edifício para o engenheiro civil, facilitando a visualização e compreensão de um sistema complexo.

Nas seções seguintes passamos a introduzir gradativamente os conceitos de POO, a notação de UML e o jargão de termos técnicos utilizados pelos especialistas da área. Portanto, nenhum material integrante deste capítulo representa a nossa contribuição individual: ele foi incluído apenas para auxiliar o leitor da astronomia a se situar neste novo terreno. Por este motivo também não ambicionamos aqui apresentar uma introdução completa e rigorosa a estes tópicos; pretendemos apenas expor o leitor aos conceitos fundamentais através de uma descrição clara e sucinta que o capacitará a compreender capítulos subseqüentes da tese. Para informações mais detalhadas, o leitor

interessado é remetido a outras fontes na literatura (Fowler & Scott 1997; Booch, Rumbaugh, & Jacobson 2000; Larman 2002).

6.2. Programação orientada a objetos e UML

Embora a definição exata do que caracteriza a POO possa ser motivo de debate entre os teóricos, na prática dizemos que uma linguagem é orientada a objetos se ela suporta os seguintes conceitos: *abstração*[▷], *encapsulamento*, *herança* e *polimorfismo*. Passamos agora a estudar cada um deles em pormenor.

Nos exemplos que se seguem apresentamos os códigos-fonte em Java (TutorialJava; Eckel 2003), que acreditamos ser uma linguagem ideal para introduzir os conceitos de POO devido à sua simplicidade.

6.2.1. Abstração

A abstração consiste na possibilidade de estender a linguagem de programação com novos tipos de dados, além daqueles já pré-definidos (inteiro, real, booleano, caractere, etc.). Estes tipos novos passam a representar diretamente em *software* os objetos do mundo real.

Para exemplificar, consideramos um aplicativo simples de desenho que pinta pontos na tela. As características essenciais de cada ponto são as suas coordenadas *x* e *y* relativas a um dos cantos da tela. Essas variáveis são convenientemente agrupadas em uma nova *classe* de objetos chamada **`Ponto`**, o que em código Java se traduziria como:

```
public class Ponto {
    public float x;
    public float y;
}
```

Podemos então definir duas variáveis deste novo tipo de dados, `ponto1` e `ponto2`, que serão *objetos* representantes da *classe* de objetos **`Ponto`**. Para que estes objetos correspondam aos pontos (0,0;0,0) e (1,0;1,0) na tela, podemos atribuir valores às suas componentes individuais usando a notação com um ponto:

```
ponto1.x = 0.0F;
ponto1.y = 0.0F;

ponto2.x = 1.0F;
ponto2.y = 1.0F;
```

[▷] Termos técnicos introduzidos pela primeira vez figurarão em itálico.

Porém antes é preciso declarar estas duas novas variáveis:

```
Ponto ponto1 = new Ponto();
Ponto ponto2 = new Ponto();
```

O comando `new` faz com que seja alocada a quantidade de memória necessária para comportar um novo objeto da classe **`Ponto`**, e associa esta região da memória à variável declarada. Um objeto de uma classe qualquer pode ser também referido como uma *instância* daquela classe, e com o ato de criar um novo objeto estamos *instanciando* um objeto da classe.

A classificação dos conceitos do mundo real em categorias é inerente ao ser humano, que vem exercitando esta habilidade pelo menos desde os tempos de Aristóteles, que agrupava os seres vivos nas classes de peixes, árvores, etc.

6.2.2. Encapsulamento

No exemplo anterior vimos que é possível reunir todos os dados característicos de uma dada abstração dentro de uma mesma unidade lógica do programa, a qual passa a representar em *software* o objeto do mundo real. A POO permite encapsular dentro das classes não apenas variáveis, mas também as rotinas que modificam os valores dessas variáveis.

Consideremos o código Java na Figura 6.1, à direita. A classe **`Dado`** representa em *software* um dado no mundo real. A única característica essencial de um dado do ponto de vista da nossa aplicação é o valor mostrado em sua face apontada para cima, armazenado em `valorFace`. Uma rotina que modifica o valor desta variável seria a rotina `jogar()`, que simplesmente a atribuiria um valor aleatório de 1 a 6. Esta rotina está embutida dentro da própria classe, e para invocá-la enviamos uma *mensagem* a um objeto da classe Dado da seguinte maneira:

```
Dado meudado = new Dado();
int numero;

meudado.jogar();
numero = meudado.getValorFace();
```

Aqui criamos um objeto `meudado` pertencente à classe **`Dado`**, e alteramos o valor de sua variável `meudado.valorFace` mandando-lhe a mensagem `meudado.jogar()`. A seguir, recuperamos o valor desta variável para a variável local `numero` mandando-lhe uma nova mensagem, `meudado.getValorFace()`. Note que a variável `valorFace` é declarada como privada, de modo que, diferentemente da classe **`Ponto`** vista anteriormente, ela não pode ser acessada diretamente e tal tentativa causará um erro em tempo de compilação:

```
numero = meudado.valorFace;  // ilegal!
```

Portanto, seu valor só pode ser alterado ou acessado indiretamente através de mensagens enviadas ao objeto `meudado`.

Uma rotina particularmente especial e única em cada classe é denominada *construtor*, e é invocada automaticamente sempre que um novo objeto é instanciado. Por exemplo, a listagem abaixo incrementa a classe **`Ponto`** com um construtor encarregado de inicializar as suas variáveis:

```
public class Ponto {

    public float x;
    public float y;

    // construtor:
    public Ponto(float abscissa, float ordenada) {
        x = abscissa;
        y = ordenada;
    }

    public void desenhe() {
        // .... código ....
    }

}
```

Listagem 6.1: A classe `Ponto`.

Em Java o construtor é declarado sempre com o mesmo nome da classe.

Podemos agora criar os objetos referentes aos pontos (0,0;0,0) e (1,0;1,0) vistos anteriormente da seguinte forma:

```
Ponto ponto1 = new Ponto(0.0F,0.0F);
Ponto ponto2 = new Ponto(1.0F,1.0F);
```

Note que aproveitamos também para incluir a rotina `desenhe()` para pintar o ponto na tela. Na versão anterior da classe **`Ponto`** não tivemos que nos preocupar com o construtor, pois o compilador Java automaticamente se encarrega de criar um construtor sem parâmetros, caso o programador não especifique um.

Analogamente, existe também uma rotina especial denominada *destrutor*, a qual é invocada automaticamente sempre que um objeto é destruído. Em Java não é necessário destruir explicitamente os objetos, pois a tarefa de recuperar a memória ocupada por objetos descartados fica a cargo de um "coletor de lixo".

A abordagem da POO de encapsular variáveis e rotinas que alteram essas variáveis dentro de uma mesma entidade em *software* favorece a criação de códigos mais robustos e fáceis de manter. Isso porque se futuramente decidirmos alterar a estrutura da classe **`Dado`** as mudanças ficarão confinadas apenas àqueles trechos do código, e não afetarão o restante do programa.

Outra vantagem decorrente da POO é a elaboração de códigos mais fáceis de entender, pois os objetos em *software* refletem diretamente as abstrações representadas pelos objetos do mundo real. Assim, eliminamos a ginástica mental de mapear o espaço de problemas no mundo real para o espaço de soluções em nosso código (Figura 6.1).

Em UML uma classe é representada como um retângulo com três compartimentos, contendo o nome da classe (em negrito), seus *atributos* (*i.e.*, suas variáveis) e seus *métodos* (as rotinas), como mostra a Figura 6.1 . Os dois últimos compartimentos podem opcionalmente ser omitidos.

Através da UML podemos especificar o projeto de nosso *software* orientado a objetos de uma maneira independente de qualquer linguagem de programação específica. Assim, partindo da classe **`Dado`** em UML na Figura 6.1, poderíamos implementar o código em Java, C++, C#, Object Pascal, Smalltalk ou qualquer outra linguagem orientada a objetos.

As ferramentas utilizadas para desenhar os diagramas em UML ainda podem se encarregar de gerar automaticamente toda a parte mecânica do código, poupando-nos de erros e nos liberando para nos ocuparmos unicamente da parte criativa de programar. Em nosso exemplo, todo o código da Figura 6.1 foi criado automaticamente pela ferramenta CASE (de *Computer-Aided Software Engineering*), restando ao programador tão somente completar o corpo do método `jogar()`.

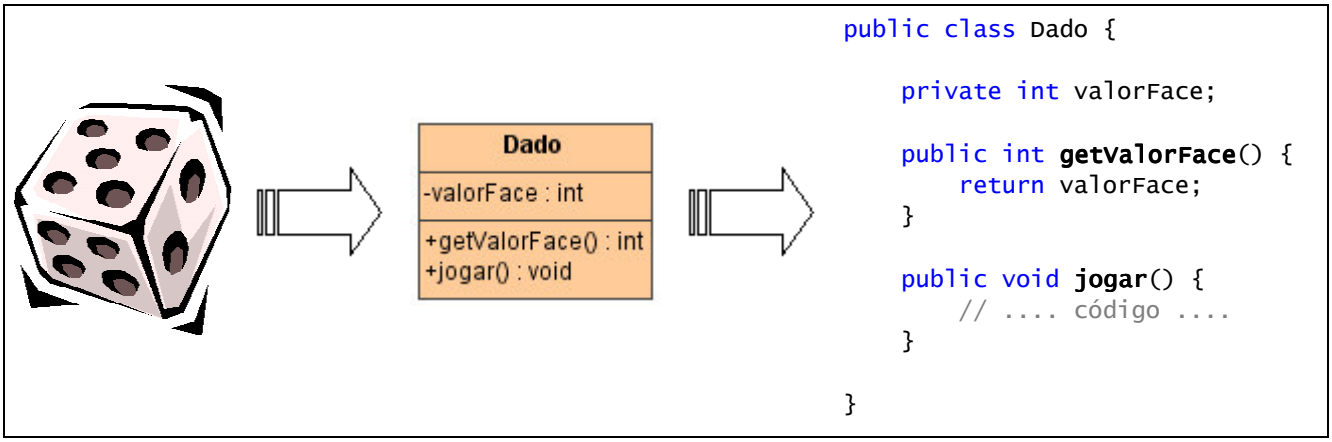


Figura 6.1: Modelagem do mundo real para UML para o *software* orientado a objetos.

6.2.3. Herança

Suponhamos que desejamos criar uma nova classe, porém muito semelhante a uma outra já existente e com funcionalidades também similares. Em POO existe um mecanismo para reaproveitar a classe original para que não seja preciso criar todo o código da nova classe desde a estaca zero, denominado *herança*.

Na listagem abaixo estendemos a classe **`Ponto`** para criar uma nova classe **`PontoColorido`**. Na definição de **`PontoColorido`** é preciso apenas incluir o novo atributo `cor`, pois os atributos `x` e `y`

são automaticamente *herdados* da classe **Ponto**. A classe original denomina-se *classe ancestral*, *classe-pai* ou *superclasse*; já a nova classe é denominada *classe-descendente*, *classe-filha* ou *subclasse*.

```
public class PontoColorido extends Ponto {

    public Cor cor;

    // construtor:
    public PontoColorido(float abscissa, float ordenada, Cor umaCor) {
        super(abscissa,ordenada);  // chama o construtor da classe ancestral
        cor = umaCor;
    }

    // o método desenhe precisa ser reescrito:
    public void desenhe() {
        // .... código ....
    }

}
```

Listagem 6.2: A classe `PontoColorido`.

Além dos atributos, a subclasse também herda todos os métodos da superclasse. Em nosso exemplo a classe **`PontoColorido`** herda o método `desenhe()` da classe **`Ponto`**, o que no entanto não é apropriado, pois agora desejamos que este método realize a tarefa mais específica de desenhar um ponto colorido na tela. Nesta situação podemos redefinir o método `desenhe()` na subclasse para que ele *se sobreponha*[▷] ao método da superclasse.

Em UML a herança é indicada por uma seta como a da Figura 6.2 . Os atributos da superclasse não são repetidos na subclasse, pois estão implícitos pela relação de herança. A repetição de um método na subclasse serve para indicar que o método da superclasse foi sobreposto.

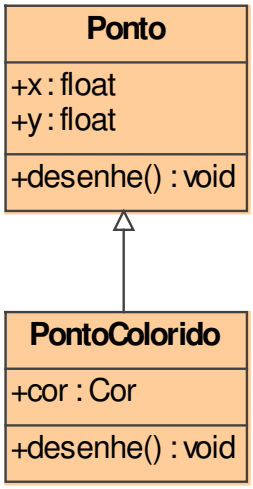


Figura 6.2: Herança.

[▷] em inglês, *override*.

A herança não apenas facilita a criação de classes relacionadas, mas também evita a duplicação de código. Isso facilita a tarefa de manutenção do nosso programa, pois se futuramente alterarmos os métodos da superclasse as mudanças automaticamente se estenderão para todas as subclasses.

É possível utilizar nossas próprias classes na construção de outras classes mais complexas, como deve ter notado o leitor mais atento ao passar pela Listagem 6.2 . A classe **`PontoColorido`** inclui como atributo um objeto da classe **`Cor`**[☞], cujo código não foi apresentado mas que simplesmente conteria os atributos `r`, `g` e `b` especificando os valores inteiros RGB que compõem a cor, juntamente com alguns métodos de acesso. De fato esta é uma maneira mais comum do que herança de reutilizar o código de outras classes, que é simplesmente juntando-as às classes novas.

A Figura 6.3 introduz o conceito de *associação* em UML, que é indicada por uma reta ligando as classes. Os números se referem à *cardinalidade* ou à *multiplicidade* da associação: em nosso exemplo cada instância de **`PontoColorido`** está associada a apenas uma instância de **`Cor`**, sendo que cada instância de **`Cor`** pode estar associada a várias instâncias de **`PontoColorido`** (outras possibilidades são mostradas na Tabela 6.1). O rótulo à direita é denominado o *papel* da associação, e deve ser interpretado como "a classe **`Cor`** representa o papel cor em relação à **`PontoColorido`**". Ou seja, o papel corresponde apenas ao nome do atributo `cor` da classe **`PontoColorido`**.

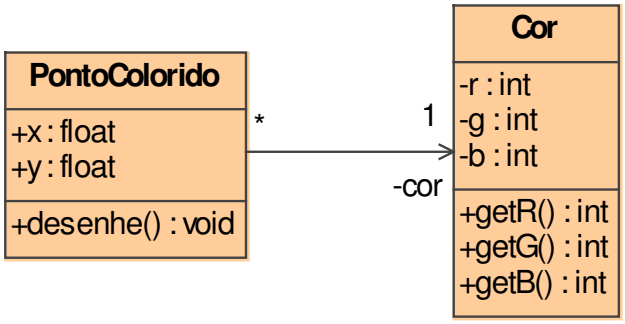


Figura 6.3: Associação entre classes.

*	de 0 a infinito
1..*	de 1 a infinito
4	exatamente 4
2, 4	2 ou 4
0..4, 6..*	qualquer, exceto 5

Tabela 6.1: Exemplos de multiplicidade em UML.

[☞] Note que Java diferencia maiúsculas e minúsculas, de modo que podemos ter um objeto `cor` da classe **`Cor`**.

A seta no extremo direito da associação diz respeito à *navegabilidade*. Em nosso caso a classe `PontoColorido` pode enviar mensagens à classe `Cor` por possuir uma referência a uma instância desta, sendo que o oposto não ocorre. Se não for indicada seta alguma teremos então que a associação será bidirecional.

As associações nos diagramas de classes indicam os caminhos pelos quais fluem as mensagens entre os objetos. Na Figura 6.4 temos uma instância de `PontoColorido` enviando uma mensagem à uma instância de `Cor`. É através das trocas de mensagens que as classes colaboram entre si para resolver as tarefas de nossa aplicação.

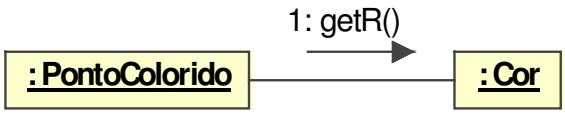


Figura 6.4: Uma mensagem sendo enviada.

6.2.4. Polimorfismo

Vamos agora sofisticar o nosso aplicativo de desenho e permitir que ele manipule formas geométricas na tela, tais como triângulos, quadrados, círculos, etc. Neste caso poderíamos ter uma hierarquia de classes como a mostrada na Figura 6.5 .

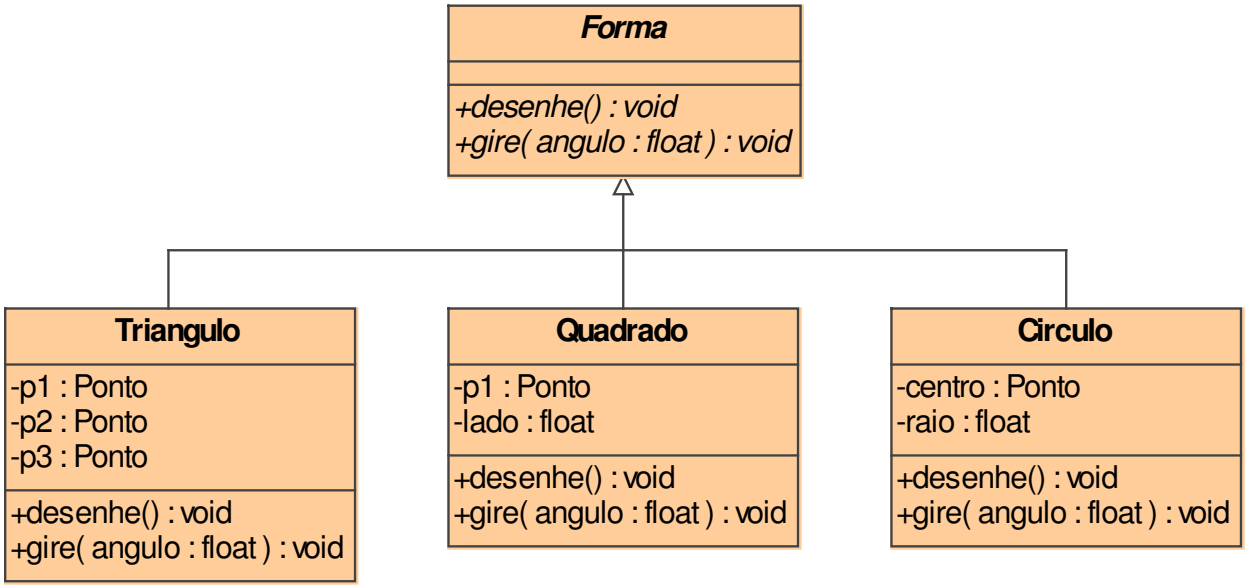


Figura 6.5: Hierarquia de classes em um aplicativo de desenho.

A implementação em Java da classe `Forma` será:

```
public abstract class Forma {
    public abstract void desenhe();
    public abstract void gire(float angulo);
}
```

Como podemos notar, o código da classe **`Forma`** não especifica a implementação dos métodos `desenhe()` ou `gire()`. Por este motivo não é possível instanciá-la diretamente, e a sua única função é a de servir como base para construção de outras classes através de herança. A classe é então denominada uma *classe abstrata*, e os métodos não implementados são *métodos abstratos*. As subclasses de uma classe abstrata deverão implementar todos os seus métodos abstratos, do contrário serão também classes abstratas. Uma classe que implementa todos os seus métodos é dita uma *classe concreta*. Classes e métodos abstratos são indicados em itálico em UML.

Todas as subclasses de **`Forma`** implementam à sua maneira os seus métodos abstratos. A classe **`Circulo`**, por exemplo, incluirá uma implementação vazia para o método `gire()`. Cada forma geométrica conterá atributos que lhe são específicos: a classe **`Triangulo`**, por exemplo, conterá os pontos que compõem os vértices do triângulo.

Na Figura 6.6 mostramos a associação entre as classes **`Triangulo`** e **`Ponto`**. Este é um exemplo de uma associação conhecida como *agregação*, a qual descreve uma relação todo-parte, como os dedos de uma mão, os pneus de um carro, etc. A agregação é indicada em UML como um losango vazio na extremidade da associação que representa o todo.

Em nosso exemplo temos a forma mais forte de agregação chamada *agregação por composição*, e indica que uma dada instância de **`Ponto`** está associada a uma única instância de **`Triangulo`**. A agregação por composição é representada por um losango cheio, e a multiplicidade no extremo do todo da associação costuma ser omitida, pois está assumida como 1.

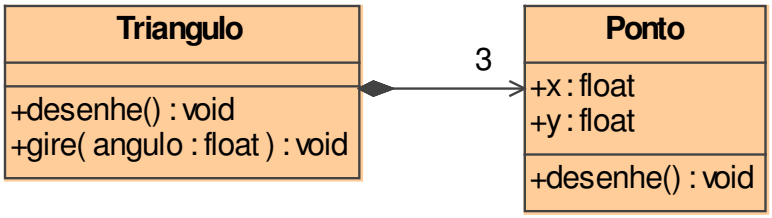


Figura 6.6: Agregação por composição.

Ocorre porém um problema com o nosso aplicativo de desenho: não há como saber de antemão qual será a forma escolhida para ser desenhada na tela, pois esta só será determinada pelo usuário em tempo de execução.

A solução para problemas deste tipo é fornecida pelo conceito de POO denominado *polimorfismo*, segundo o qual é possível inserir um objeto de uma classe descendente em qualquer contexto que espere um objeto da classe ancestral. Assim o nosso código pode ser compilado para lidar com uma forma genérica através da classe **`Forma`**, e em tempo de execução receber uma instância de qualquer uma de suas subclasses, **`Triangulo`**, **`Quadrado`** ou **`Circulo`**.

O código a seguir manipula as subclasses pela superclasse:

```
Forma forma = new Triangulo ( new Ponto(0.0F,0.0F), new Ponto(1.0F,1.0F),
                              new Ponto(2.0F,0.0F) ) ;
forma.desenhe();
forma = new Quadrado ( new Ponto(0.0F,0.0F), 1.0F ) ;
forma.desenhe();
forma = new Circulo ( new Ponto(1.0F,1.0F), 1.0F ) ;
forma.desenhe();
```

Em todos os casos a versão correta do método `desenhe()` é invocada.

Outra maneira de manipular objetos polimorficamente é definir uma *interface*, que é uma espécie de classe abstrata pura, ou seja, sem nenhum método implementado. Vamos definir por exemplo a interface **`Desenhavel`**:

```
public interface Desenhavel {
    void desenhe();
}
```

Podemos agora fazer com que as classes **`Ponto`** e **`Forma`** *implementem* a interface:

```
public class Ponto implements Desenhavel {
    // .... código ....
}

public abstract class Forma implements Desenhavel {
    // .... código ....
}
```

Caso o compilador não encontre um método `desenhe()` na definição das classes, uma mensagem de erro será emitida.

Podemos então manipular as classes **`Ponto`** e **`Forma`** através da interface **`Desenhavel`**:

```
Ponto ponto = new Ponto(0.0F,0.0F);
Forma forma = new Circulo ( new Ponto(1.0F,1.0F), 1.0F ) ;
Desenhavel coisa = ponto;
coisa.desenhe();
coisa = forma;
coisa.desenhe();
```

Entretanto ficamos restritos unicamente aos métodos especificados pela interface, de modo que o código a seguir causará um erro de compilação:

```
coisa = forma;
coisa.gire(90.0F);  // ilegal!
```

Percebemos então que o que uma interface faz é justamente prover uma interface de acesso às classes. Tecnicamente as interfaces são úteis para manipular polimorficamente classes não-relacionadas, sem a necessidade de impor uma hierarquia de herança entre elas.

Em UML as interfaces são representadas de forma semelhante às classes, e a relação de implementação é indicada por uma seta tracejada como mostra a Figura 6.7 .

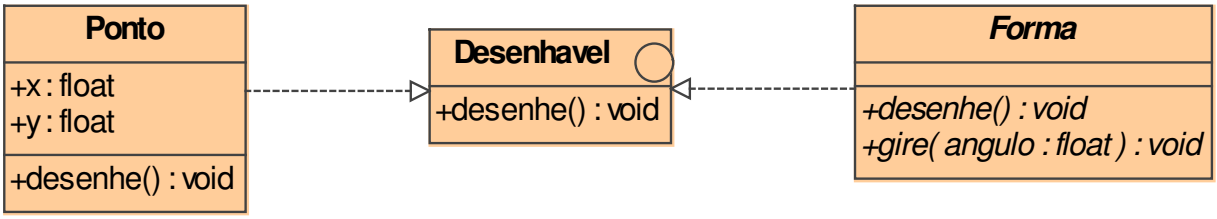


Figura 6.7: A relação de implementação.

6.2.5. Outros conceitos

Nesta seção reunimos mais alguns conceitos para completar a nossa discussão de POO e UML, e que não foram cobertos nas seções anteriores.

Algumas vezes desejamos que o valor de um determinado atributo seja compartilhado por todas as instâncias da classe. Suponhamos que desejamos contabilizar o número de objetos alocados de uma determinada classe. Neste caso poderíamos armazenar o número de instâncias em um atributo `nInstancias`, o qual seria acrescido de 1 no construtor e diminuído de 1 no destrutor[1], como mostra a Listagem 6.3 .

Dizemos que `nInstancias` é um *atributo estático*[2], e o seu valor será o mesmo em todas as instâncias de **`Classe`**. Assim, após a execução de:

```
Classe objeto1 = new Classe(5);
Classe objeto2 = new Classe(6);
```

teremos `objeto1.nInstancias` = 2 , `objeto1.atributo` = 5 , `objeto2.nInstancias` = 2 e `objeto2.atributo` = 6 .

Métodos estáticos são aqueles que acessam unicamente os atributos estáticos da classe, e não podem de forma alguma acessar seus atributos não-estáticos.

Atributos e métodos estáticos podem ser também acessados diretamente pela classe, sem a necessidade de obter previamente uma instância da mesma:

```
int i = Classe.getNInstancias();
```

---

[1] Como em Java não é preciso destruir explicitamente os objetos, devemos chamar o método `finish()` antes de descartar as instâncias de **`Classe`**. O mesmo efeito não pode ser atingido sobrecarregando-se o método `finalize()`, uma vez que a máquina virtual Java poderá executá-lo num momento posterior que julgar mais conveniente.

[2] *Estático* é um termo do jargão de Java, o qual preferimos empregar ao invés do equivalente em UML que seria *no escopo da classe*.

, sendo equivalente a:

```
int i = objeto1.getNInstancias();
```

Acreditamos que a primeira alternativa seja mais adequada, por enfatizar que o método é estático.

Em UML atributos e métodos estáticos são sublinhados, como mostra a Figura 6.8 .

```
public class Classe {

    // um atributo estático:
    private static int nInstancias=0;

    // um atributo comum (no escopo da instância):
    private int atributo;

    // construtor:
    public Classe(int valor) {
        nInstancias++;
        atributo = valor;
    }

    // "destrutor":
    public void finish() {
        nInstancias--;
    }

    // um método estático:
    public static int getNInstancias() {
        return nInstancias;
    }

    // um método comum (no escopo da instância):
    public int getAtributo() {
        return atributo;
    }

}
```

Listagem 6.3: Atributos e métodos estáticos.

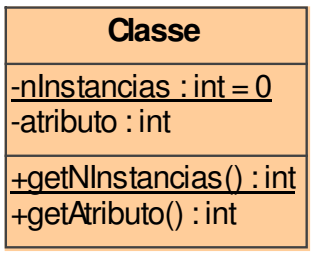


Figura 6.8: Atributos e métodos estáticos.

Outro conceito interessante é o de *pacotes*, que permitem reunir classes relacionadas para compor bibliotecas inteiras. Para criar um pacote `desenho` contendo todas as classes de nosso aplicativo de desenho, basta adicionar a seguinte linha antes da definição de cada classe:

```
package desenho;
```

E então sempre que desejarmos utilizar as classes deste pacote tudo o que temos a fazer é incluir:

```
import desenho.*;
```

Pacotes também são úteis para organizar as classes em espaços de nomes diferentes a fim de evitar conflitos de nomes. Suponhamos que estamos utilizando também uma outra biblioteca de terceiros que também contenha uma classe **`Circulo`**. Neste caso podemos desfazer a ambigüidade nos referindo à nossa classe como `desenho.Circulo` .

Em UML um pacote é representado como uma pasta, como mostra a Figura 6.9 .

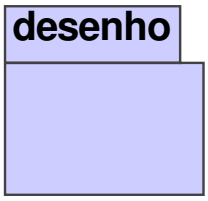


Figura 6.9: Um pacote.

Nas seções anteriores vimos que a *visibilidade* de atributos e métodos de uma classe pode ser especificada como pública ou privada. Ao projetarmos as nossas classes devemos procurar atribuir a visibilidade privada ao maior número possível de atributos ou métodos, pois ao escondê-los do mundo exterior fica mais fácil alterá-los posteriormente sem afetar outras regiões de nosso código. Existem ainda dois outros tipos de *especificadores de acesso*[1]: *protegido*, que torna o atributo ou método visível somente às subclasses da classe, e *pacote*, que os torna visíveis a todas as outras classes incluídas no mesmo pacote. A visibilidade pública torna os atributos e métodos acessíveis a qualquer outra classe sem restrições, ao passo que a visibilidade privada é a forma mais rígida de escondê-los, tornando-os visíveis apenas dentro da própria classe. A Tabela 6.2 mostra como são indicados os vários especificadores de acesso em UML.

Com isso encerramos a nossa discussão dos princípios básicos de POO e da notação em UML. Cabe salientar que foi apresentada apenas a notação relativa aos *diagramas de classes*, sendo que a UML é uma linguagem muito mais abrangente que fornece nove tipos de diagramas os quais

---

[1] Este termo é do jargão de Java.

permitem modelar diferentes aspectos de nosso *software*. Os outros tipos de diagramas não figurarão com muita freqüência neste trabalho, sendo que apenas eventualmente incluiremos um ou outro sem aviso prévio, como fizemos na Figura 6.4, que é um exemplo de um *diagrama de colaboração*. Os diagramas nestes casos, porém, serão relativamente simples e intuitivos.

+	público
-	privado
#	protegido
~	pacote

Tabela 6.2: Especificadores de acesso.

6.3. Padrões de projeto

O paradigma da orientação a objetos é uma ferramenta poderosa para assegurar a qualidade de códigos extremamente complexos. Porém o simples fato de utilizarmos os conceitos fundamentais de POO em nosso programa por si só não garante que o produto final será um código fácil de manter, adaptar, estender, entender e dividir com colaboradores. Como diz o provérbio em inglês: "possuir um martelo não faz de ninguém um arquiteto". Ou seja, é preciso saber utilizar a ferramenta com habilidade, e para realizar todos os benefícios da POO é necessário respeitar certas regras gerais ao projetarmos o nosso código.

Muitos destes princípios são elementares, como "esconder o máximo de informação em atributos e métodos privados". Entretanto o tópico de análise e projeto orientado a objetos é extenso e não pode ser adequadamente introduzido neste trabalho, pelo que remetemos o leitor a outras fontes na literatura (Larman 2002).

Certos problemas recorrentes que surgem com muita freqüência na prática já possuem soluções elegantes desenvolvidas. Estas soluções são catalogadas na forma de *padrões*, e capturam o conhecimento de programadores mais experientes que desenvolveram soluções gerais que podem ser reutilizadas por outros programadores.

Por exemplo, muitas vezes é necessário fazer com que uma determinada classe possa ser instanciada uma única vez, e que todos os pontos da aplicação acessem esta mesma instância. A solução para este problema é dada pelo padrão *Singleton* (Gamma et al. 1995), o qual é implementado em Java na Listagem 6.4 . No padrão *Singleton* não permitimos que outras classes acessem o construtor, e a única maneira de obter uma instância de nossa classe é através do método estático `getInstance()`, que retorna sempre a mesma instância armazenada no atributo estático `singletonInstance`. Como classes são geralmente acessíveis, todas as outras partes do programa podem facilmente obter acesso à esta instância.

```
// O padrão Singleton:
public class Singleton {

    // instância singleton:
    private static Singleton singletonInstance;

    // método de acesso à instância:
    public static Singleton getInstance() {
        if ( singletonInstance == null ) {
            singletonInstance = new Singleton();
        }
        return singletonInstance;
    }

    // construtor não é acessível:
    protected Singleton() {
    }

}
```

Listagem 6.4: O padrão *Singleton* em Java.

Não há motivos para supor que os padrões identificados em outras áreas não sejam aplicáveis também em códigos científicos.

Muitas aplicações científicas realizam simulações numéricas dividindo o espaço numa grade de células onde as condições físicas são supostas como constantes. Para fazer uso da memória disponível de maneira eficiente, é necessário que a grade de regiões espaciais seja refinada somente nos setores onde isso é realmente necessário. Em duas dimensões poderíamos ter então a situação ilustrada na Figura 6.10 .

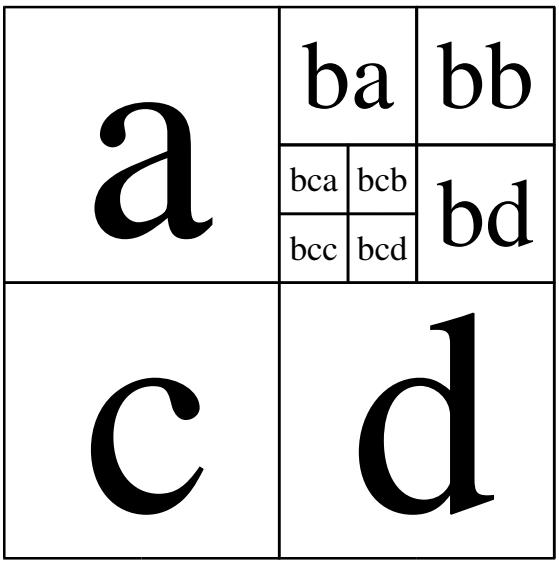


Figura 6.10: Resolvendo adequadamente a grade de regiões espaciais.

Códigos deste tipo podem ser implementados com o padrão *Composite* (Gamma, et al. 1995), mostrado na Figura 6.11 . Uma dada célula pode ser de dois tipos: uma "célula concreta" ou uma "célula composta". Esta última seria apenas um contêiner para as sub-regiões de uma determinada região, as quais por suas vez podem ser células concretas ou mesmo outras células compostas, recursivamente. Assim, a grade de regiões espaciais da Figura 6.10 poderia em tempo de execução ser representada pelo diagrama de objetos na Figura 6.12 .

Ao longo de nosso trabalho utilizaremos padrões de projeto com bastante freqüência.

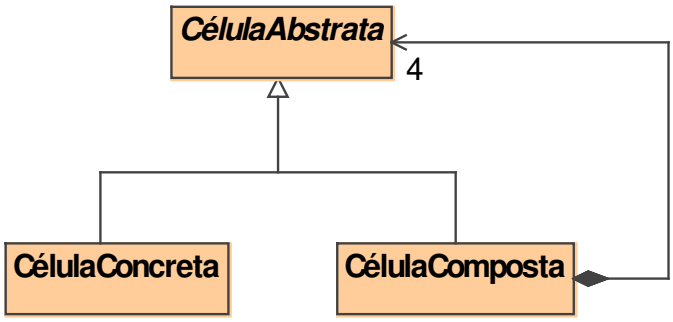


Figura 6.11: O padrão *Composite*.

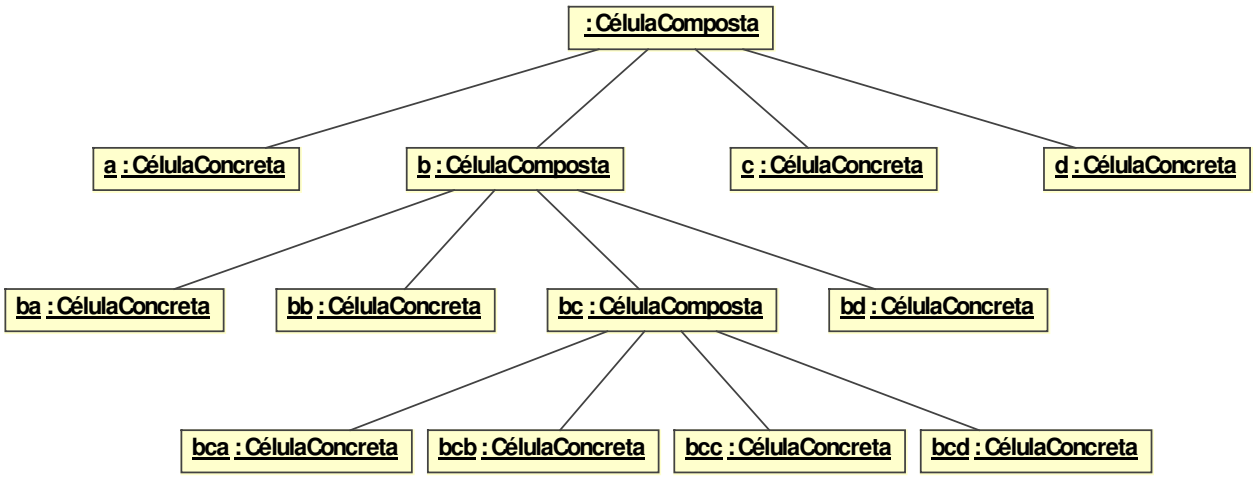


Figura 6.12: Instâncias das células da grade de regiões espaciais.

Referências

Booch, G., Rumbaugh, J., & Jacobson, I. 2000, *"UML - Guia do Usuário"* (Campus)

Eckel, B. 2003, *"Thinking in Java"*, 3ª ed. (Upper Saddle River, NJ: Prentice Hall PTR),(disponível gratuitamente em http://www.mindview.net)

Ferland, G. J. 1996, *"Hazy, a Brief Introduction to Cloudy"*, Internal Report (University of Kentucky Department of Physics and Astronomy)

Fowler, M., & Scott, K. 1997, *"UML Distilled"* (Addison-Wesley)

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. 1995, *"Design Patterns"* (Addison-Wesley)

Larman, C. 2002, *"Applying UML and patterns"*, 2ª ed. (Upper Saddle River, NJ: Prentice Hall PTR)

Moore, G. E. 1965, Electronics, 38, 114

TutorialJava, Tutorial Java da Sun: http://java.sun.com/docs/books/tutorial/index.html

7. Fortran Orientado a Objetos

7.1. Introdução

O Fortran é uma das linguagens de alto nível mais antigas que existe. Nos áureos tempos das décadas de 50 e 60, os programadores lidavam diretamente com as instruções do processador da máquina na qual estavam trabalhando, uma tarefa extremamente tediosa e difícil. Foi então que uma equipe liderada por John Backus na IBM desenvolveu uma linguagem de mais alto nível que poderia ser facilmente entendida e que possibilitaria codificar as fórmulas numéricas de uma maneira mais próxima da notação matemática. Esta nova linguagem foi batizada de Fortran, como uma contração de *FORmula TRANslation*.

A criação do Fortran gerou um salto na produtividade dos programadores, permitindo que os códigos fossem desenvolvidos muito mais rapidamente, às custas apenas de uma pequena perda de desempenho em tempo de execução.

A nova linguagem foi extremamente bem sucedida e acabou sendo adotada pela maioria dos programadores, o que acabou levando também à proliferação de inúmeros dialetos de Fortran. Com o intuito de facilitar o intercâmbio de programas entre os pesquisadores é que foi produzido então o primeiro padrão de uma linguagem de programação, hoje conhecido como Fortran 66. Desde então o padrão foi revisado outras três vezes, em 1978, 1991 e 1997, levando ao Fortran 77, Fortran 90 e Fortran 95. Destes, o Fortran 90 foi uma grande revisão, ao passo que as mudanças introduzidas com o Fortran 77 e Fortran 95 foram relativamente menores.

Provavelmente o maior benefício de se programar numa linguagem de alto nível seja o de permitir nos liberarmos do encargo de nos preocupar com a implementação do nosso código na máquina, e nos concentramos mais em nosso problema em si. Porém, desde o surgimento do Fortran nossos códigos cresceram muito em termos de complexidade, e, como vimos no capítulo anterior, os cientistas já não têm sido tão produtivos empregando as ferramentas de desenvolvimento tradicionais. A adoção do paradigma da POO seria mais um passo adiante para lidar com códigos muito complexos, pois nos possibilitaria programar numa linguagem mais abstrata e próxima dos conceitos do nosso problema, sem necessitarmos nos preocupar em como representá-lo nas estruturas da máquina.

O código de fotoionização Aangaba, o qual utilizaremos mais adiante em nossa aplicação, foi escrito em Fortran 77. Por esse motivo elegemos o Fortran 90/95 como a nossa linguagem de

programação para implementar o banco de dados atômicos descrito no capítulo 3, e integrá-lo com sucesso ao Aangaba. A fim de poder utilizar o ferramental introduzido no capítulo anterior precisamos saber até que ponto o Fortran moderno suporta os conceitos chave da POO.

Como veremos na seção 7.2, é possível realizar POO em Fortran, embora algumas funcionalidades fundamentais estejam ausentes. Porém, praticamente todos os conceitos básicos de POO podem ser adequadamente emulados às custas de um esforço adicional. Atualmente encontra-se em processo de preparação um novo padrão do Fortran, o qual, assim como o Fortran 90, será uma grande extensão sobre as versões anteriores. O novo padrão do Fortran incluirá suporte direto à POO, de modo que podemos encarar o esforço adicional em utilizar POO no Fortran 90/95 como um investimento no futuro. Na seção 7.3 apresentamos uma prévia do que podemos esperar deste novo padrão, e como ele irá resolver os problemas relacionados à implementação de POO em seu antecessor. Finalmente, nossas conclusões gerais são apresentadas na seção 7.4 .

7.2. Emulando POO em Fortran 90/95

O Fortran 90 foi um desenvolvimento significativo em relação ao padrão anterior, o Fortran 77, e introduziu características como: alocação dinâmica de memória, sobrecarga de rotinas e operadores, a linguagem de *arrays*, criação de tipos de dados, novas estruturas de controle de execução como o comando `case`, entre outras. Estas mudanças visaram incorporar ao Fortran características já experimentadas e testadas em outras linguagens como Pascal, C e C++, e o tornaram mais competitivo com novas linguagens que surgiram ao longo da década de 80 para as quais vinha perdendo terreno. Já o Fortran 95 foi apenas uma pequena revisão que se sucedeu alguns anos depois a fim de torná-lo compatível com o High Performance Fortran, evitando a proliferação de novos dialetos de Fortran. Para maiores detalhes sobre a linguagem Fortran, o leitor é remetido ao livro de Metcalf & Reid (1999).

Dentre as novidades introduzidas em Fortran 90/95, a linguagem de *arrays* é geralmente a mais celebrada. Ainda que esta seja extremamente útil em aplicações científicas, poucos sabem que o novo padrão da linguagem também possibilita a criação de códigos mais robustos suportando modelos de programação mais abstratos.

Embora a POO não esteja oficialmente incluída nos últimos avanços, vários autores têm demonstrado que é possível realizar POO em Fortran 90/95 (Decyk, Norton, & Szymanski 1996; Norton 1996; Norton, Decyk, & Szymanski 1996; Cary et al. 1997; Decyk, Norton, & Szymanski 1997a, 1997b; Gray & Roberts 1997; Decyk, Norton, & Szymanski 1998), emulando todos os conceitos chave sem suporte direto, incluindo os menos evidentes tais como herança e polimorfismo (Decyk, et al. 1998). Por intermédio da emulação, pôde-se preencher a lacuna existente entre o Fortran 90/95 e o Fortran 2003, permitindo a vários autores implementarem com sucesso os seus códigos orientados a objetos em aplicações da física e matemática (Dupée 1994; Norton,

Szymanski, & Decyk 1995; Norton 1996; Machiels & Deville 1997; Kim, Lee, & Martin 2000; Qiang, Ryne, & Habib 2000; Akin & Singh 2002).

O objetivo das próximas seções é apresentar o método desenvolvido pelos outros autores para realizar POO em Fortran, juntamente com algumas contribuições inéditas nesse setor frutos de nossa própria experiência em implementar nosso código neste esquema. Em todos os casos estas contribuições estarão indicadas claramente no texto.

Para atingir este objetivo, implementaremos gradativamente as classes do diagrama na Figura 7.1, as quais servirão para ilustrar os conceitos fundamentais de POO.

A classe **`Ancestral`** possui apenas um atributo no escopo da instância, `atributo1`, com seus métodos de acesso `getAtributo1()` e `setAtributo1()`. O atributo estático `nInstancias` serve para contabilizar o número de objetos alocados da classe, e é alterado pelos métodos privados `incNInstancias()` e `decNInstancias()`, e acessado pelo método público `getNInstancias()`.

A classe **`Descendente`** especializa a classe **`Ancestral`** adicionando um novo atributo no escopo da instância, bem como métodos de acesso associados.

A interface **`Funcao`** possui apenas um método, `avalie()`, e é implementada pelas classes **`Ancestral`** e **`Descendente`**. O método `avalie()` implementado em **`Ancestral`** é sobreposto por **`Descendente`**.

As classes **`ClasseAbstrata`** e **`ClasseConcreta`** servem para ilustrar a implementação de classes abstratas e também para evidenciar um problema com o mecanismo proposto pelos outros autores para emular o conceito de herança, o qual discutimos na seção 7.2.2 .

As classes da Figura 7.1 também foram implementadas em Java, e os resultados de dois programas de exemplo simples foram comparados com as saídas das implementações equivalentes em Fortran. Todos os códigos fonte, em Fortran e Java, estão anexados no apêndice C e também estão disponíveis no CD que acompanha a tese.

A emulação dos conceitos de POO em Fortran, em particular herança e polimorfismo, é uma tarefa tediosa e sujeita a erros. Por isso ao implementar aplicações razoavelmente elaboradas seguindo este método é fundamental adotarmos uma convenção de estilo consistente, o que infelizmente não foi uma preocupação da maioria dos autores. A convenção de estilo aqui apresentada foi desenvolvida com o intuito de facilitar a transição posterior do código para o Fortran 2003, procurando antecipar ao máximo possível a sintaxe proposta atualmente em processo de votação.

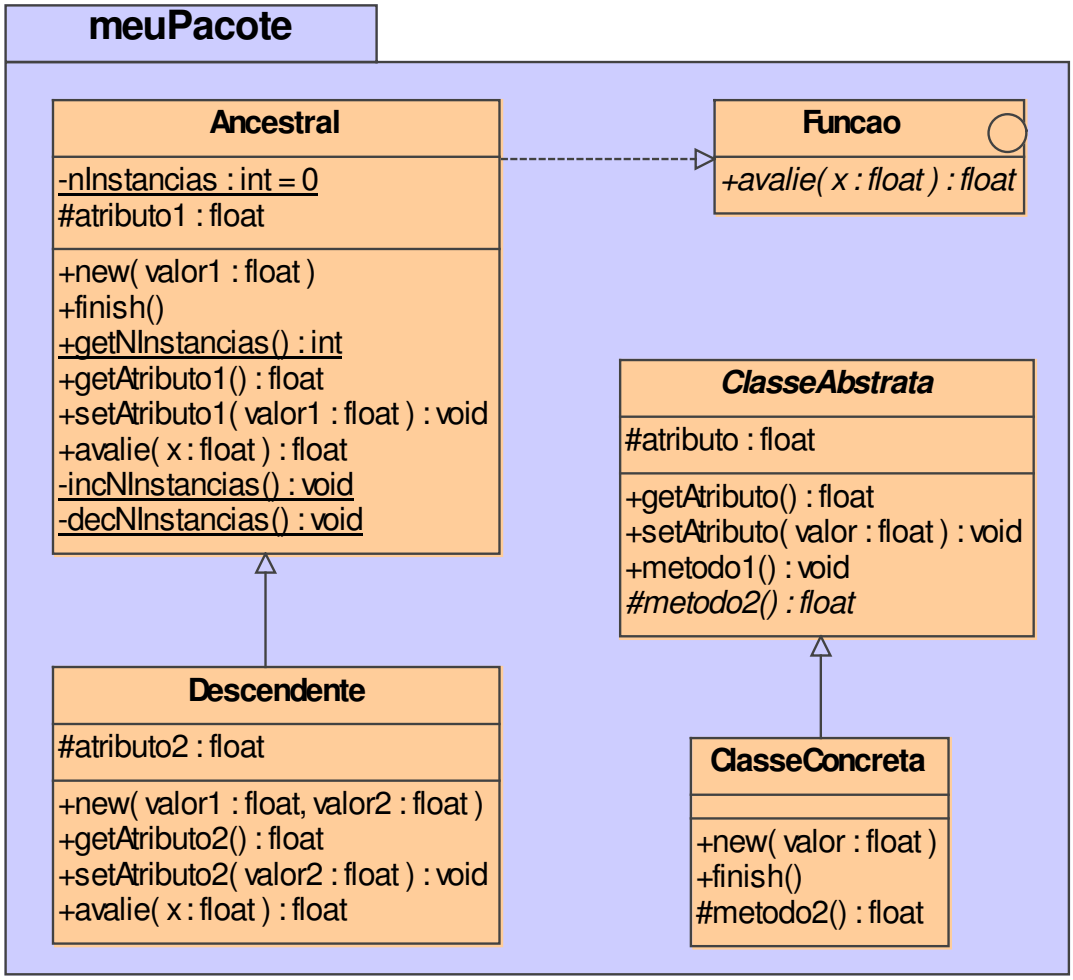


Figura 7.1: Diagrama de classes ilustrando os conceitos básicos de POO.

7.2.1. Classes

Em Fortran 95 podemos criar novos tipos de dados com o comando `type`:

```
type, public :: Ponto
    ! private
    real :: x
    real :: y
end type Ponto
```

Para criar e manipular variáveis deste novo tipo de dados a sintaxe é:

```
type(Ponto) :: ponto1, ponto2

ponto1%x = 0.0
ponto1%y = 0.0

ponto2%x = 1.0
ponto2%x = 1.0
```

Podemos notar que o Fortran utiliza o símbolo de porcentagem para acessar as componentes individuais do novo tipo, ao passo que praticamente qualquer outra linguagem utiliza um ponto. Outra diferença é que em Fortran não é possível especificar a visibilidade de cada uma das componentes individualmente: elas devem ser todas públicas ou todas privadas. No exemplo acima, para tornar todas as componentes privadas devemos descomentar a linha com o comando `private`. A sintaxe não permite a inclusão do comando `public` para enfatizar o comportamento padrão de que as componentes são públicas.

Para inicializar todas as componentes de um novo tipo podemos utilizar uma notação mais conveniente denominada *construtor de estrutura*[⚐]:

```
ponto1 = Ponto(0.0,0.0)
ponto2 = Ponto(1.0,1.0)
```

Os construtores de estruturas estão automaticamente disponíveis para os novos tipos criados pelo usuário, exceto nos casos em que as componentes são definidas como privadas.

Não é possível encapsular rotinas junto com o novo tipo de dados para compor classes, como em POO. Porém podemos obter o mesmo efeito reunindo a definição do novo tipo com as rotinas que modificam suas componentes dentro de um mesmo *módulo* em Fortran.

A Listagem 7.1 mostra a implementação em Fortran de uma classe com apenas um atributo e um método no escopo da instância, e um atributo e método estáticos.

O comando `implicit none` está sempre presente na seção inicial de qualquer bom programa em Fortran 90/95, e serve para indicar ao compilador que todas as variáveis devem ter os seus tipos explicitamente declarados[⚐⚐]. Logo a seguir temos o comando `private`, que faz com que todas as entidades presentes no módulo sejam assumidas como privadas, a não ser nos casos em que o programador explicitamente as especifique como públicas. Este é um artifício para impedir que as entidades de outros módulos utilizados pelo nosso módulo não acabem poluindo o espaço de nomes de outras unidades que venham a importar o nosso módulo. Estas duas recomendações são práticas comuns em linguagem F.

A definição da classe **`Ancestral`** inclui um único atributo no escopo da instância, `atributo1`, e para fixar o seu valor temos o método `setAtributo1`. Em Fortran um método é implementado como uma rotina que toma uma instância da classe explicitamente como um de seus parâmetros. Em nossa convenção, este parâmetro é identificado na subrotina pelo nome `this`, e sempre figura em primeiro lugar na lista de parâmetros. Para evitar conflitos de nomes com métodos homônimos de outras classes, convencionamos ainda anteceder o nome da rotina pelo

---

[⚐] Em inglês, *structure constructor*.

[⚐⚐] Isto também é conhecido como *tipagem forte*.

nome da classe, de forma que na Listagem 7.1 temos o método declarado como `Ancestral_setAtributo1()`. Por fim, para simplificar o acesso ao método podemos sobrecarregar[⚐] o nome `setAtributo1` para que ele também se refira à rotina recém-criada:

```
interface setAtributo1
    module procedure Ancestral_setAtributo1
end interface setAtributo1
```

Supondo que `meuAncestral` é uma instância da classe, podemos então enviar uma mensagem da seguinte forma:

```
call setAtributo1(meuAncestral,1.0)
```

, o que seria equivalente a, em Java:

```
meuAncestral.setAtributo1(1.0F);
```

Chamamos a atenção para a armadilha em Fortran de se definir parâmetros de rotinas que são tipos derivados como `intent(out)`, o que pode fazer com que as componentes do tipo não fixadas dentro da rotina se tornem indeterminadas na saída (Clerman 1999; Decyk & Norton 1999; Decyk, Norton, & Szymanski 1999; Decyk & Norton 2000a, 2000b). Para prevenir este problema e evitar que os valores de alguns atributos de nossos objetos se percam, devemos ter o cuidado de evitar declará-los como `intent(out)`, e ao invés disto usar sempre `intent(inout)`.

O construtor da classe é implementado por uma rotina `new`, uma vez que não é possível sobrecarregar o construtor de estrutura do tipo com uma nova função. A rotina que faz o papel de destrutor, por sua vez, é denominada `finish`. Em linguagens com suporte a POO estas rotinas seriam automaticamente chamadas ao criar ou destruir instâncias da classe, ao passo que aqui devemos chamá-las explicitamente nestas situações.

Atributos estáticos são implementados como variáveis globais no módulo, como fizemos com o atributo `nInstancias` na Listagem 7.1 . É preciso utilizar o comando `save` para fazer com que as variáveis não se tornem indefinidas ao deixarmos o escopo do módulo. A fim de evitar conflitos de nomes com atributos estáticos de outras classes, o nome do atributo é antecedido pelo nome da classe.

Em linguagens orientadas a objeto os atributos estáticos estão acessíveis também através das instâncias. Isto também pode ser conseguido em Fortran, o que não foi apontado por outros autores. Tudo o que temos a fazer é incluir a palavra-chave `target` na declaração do atributo estático, e incluir um novo atributo como um ponteiro para a variável global na definição da nossa classe:

[⚐] Em inglês, *overload*.

```
integer, target :: Ancestral_nInstancias = 0

save

type, public :: Ancestral
    private
    integer, pointer :: nInstancias => null()
    real :: atributo1
end type Ancestral
```

Como em Fortran os ponteiros só podem ser inicializados para `null()`, é preciso fixá-lo no construtor da classe:

```
subroutine Ancestral_new (this,valor1)
type(Ancestral), intent(inout) :: this
real, intent(in) :: valor1

this%nInstancias => Ancestral_nInstancias
! .... código ....

end subroutine Ancestral_new
```

Em nossa convenção, porém, preferimos acessar o atributo unicamente através da classe a fim de enfatizar a sua natureza estática.

Métodos estáticos são implementados de maneira semelhante aos métodos no escopo da instância, e a única diferença é que eles não recebem uma instância da classe com o parâmetro `this`. Na Listagem 7.1 temos o método estático `getNInstancias`, que seria chamado da seguinte maneira:

```
integer :: numero
numero = Ancestral_getNInstancias()
```

, o que seria análogo a, em Java:

```
int numero;
numero = Ancestral.getNInstancias();
```

Esta é a maneira de emular o conceito de classe em Fortran 90/95. Salientamos ainda que devemos nos restringir a definir apenas uma classe por módulo, pois do contrário seria possível a uma dada classe acessar os atributos e métodos privados de outra.

```
module meuPacote_Ancestral_Class

implicit none

! acessibilidade padrão:
private

! atributos no escopo da classe:
integer :: Ancestral_nInstancias = 0

! necessário para que as variáveis não fiquem indefinidas:
save

type, public :: Ancestral
    ! atributos no escopo da instância:
    private
    real :: atributo1
end type Ancestral

! métodos públicos:
! .... código ....
public :: Ancestral_getNInstancias, setAtributo1
! .... código ....

! interfaces para métodos no escopo da instância:

! .... código ....

interface setAtributo1
    module procedure Ancestral_setAtributo1
end interface setAtributo1

! .... código ....

contains

! .... código ....

function Ancestral_getNInstancias () result(nInstancias)
integer :: nInstancias

nInstancias = Ancestral_nInstancias

end function Ancestral_getNInstancias

subroutine Ancestral_setAtributo1 (this,valor1)
type(Ancestral), intent(inout) :: this
real, intent(in) :: valor1

this%atributo1 = valor1

end subroutine Ancestral_setAtributo1

! .... código ....

end module meuPacote_Ancestral_Class
```

Listagem 7.1: Uma "classe" em Fortran 95.
O código foi abreviado a fim de simplificar a discussão. A listagem completa está disponível no apêndice C.

7.2.2. Herança

O conceito de herança pode ser emulado em Fortran 90/95 incluindo-se na classe descendente uma componente do tipo da classe ancestral, a qual convencionamos chamar sempre de `super`. Por exemplo, para estender a classe **`Ancestral`** com um novo atributo - `atributo2` – teríamos:

```
type, public :: Descendente
    private
    type(Ancestral) :: super
    real :: atributo2
end type Descendente
```

O atributo da classe ancestral estaria então disponível como `this%super%atributo1`.

Todos os métodos da classe ancestral precisam ser reescritos, porém as novas implementações nas subclasses simplesmente delegam as tarefas para os métodos correspondentes da superclasse:

```
subroutine Descendente_setAtributo1 (this,valor1)
type(Descendente), intent(inout) :: this
real, intent(in) :: valor1

call setAtributo1(this%super,valor1)

end subroutine Descendente_setAtributo1
```

Note que o conceito de herança foi efetivamente emulado, pois não há duplicação de código. A nova implementação consiste em apenas uma linha transferindo o trabalho para a superclasse, e quaisquer alterações no método original automaticamente se refletirão no método herdado.

A notação acima também pode ser empregada para invocar o método da superclasse quando este é sobreposto na subclasse:

```
subroutine Descendente_new (this,valor1,valor2)
type(Descendente), intent(inout) :: this
real, intent(in) :: valor1, valor2

! chama o construtor da superclasse:
call new(this%super,valor1)

call setAtributo2(this,valor2)

end subroutine Descendente_new
```

A herança de métodos estáticos também pode ser emulada de maneira similar.

Com relação ao atributo estático `Ancestral_nInstancias`, este não estará disponível através da nova classe como `Descendente_nInstancias`. Infelizmente não é possível definir uma variável global no módulo da subclasse como um ponteiro que aponte para a variável

correspondente ao atributo estático da superclasse, pois em Fortran os ponteiros só podem ser inicializados para `null()`.

Temos então que a herança é emulada em Fortran agregando-se uma instância da classe ancestral à classe descendente, e implementando-se métodos semelhantes que apenas delegam o trabalho para os métodos da superclasse, como mostra a Figura 7.2 .

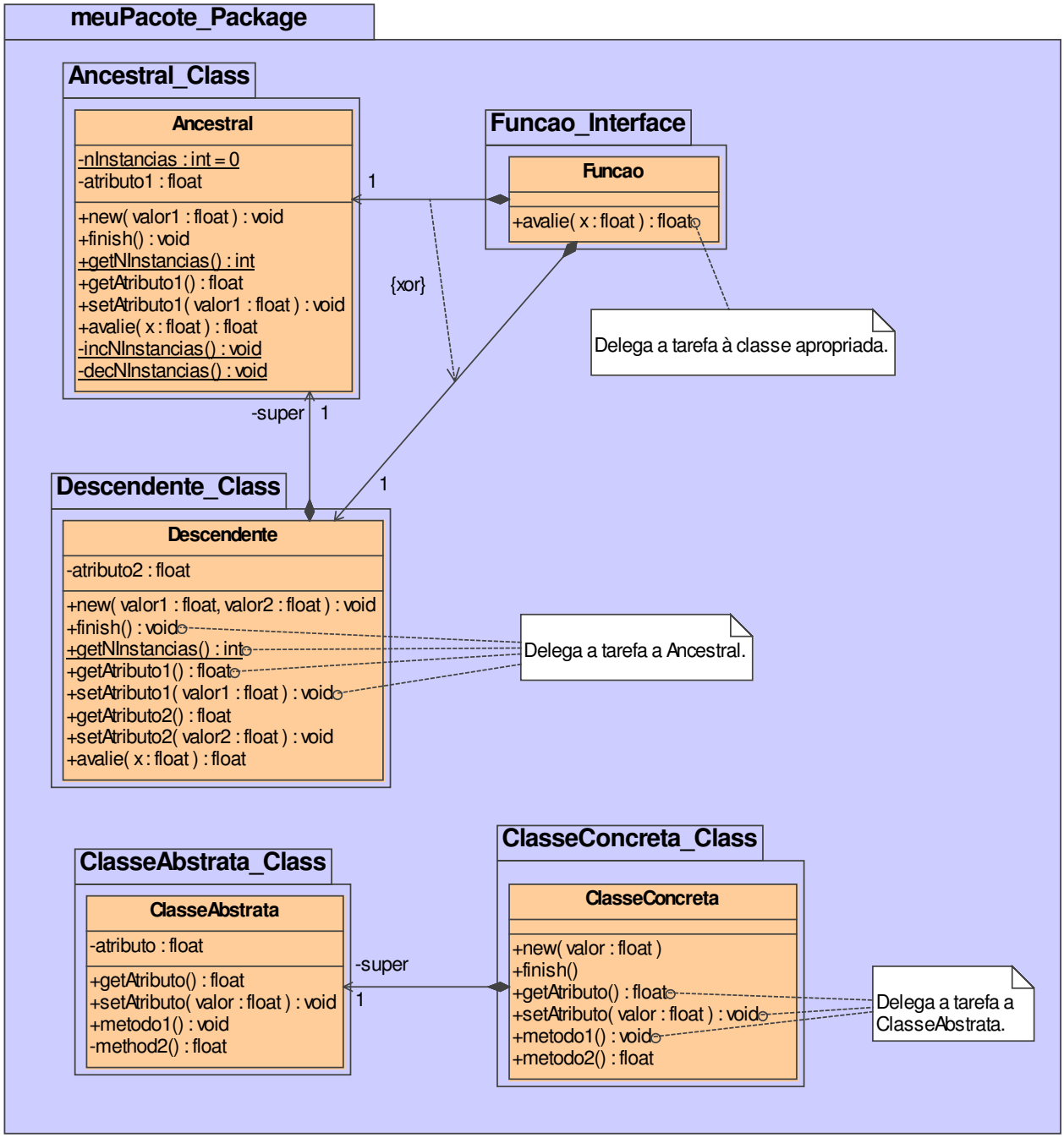


Figura 7.2: Emulando os conceitos de POO em Fortran 90/95.

Um inconveniente que não foi apontado por outros autores diz respeito à visibilidade protegida, pois os únicos especificadores de acesso disponíveis em Fortran são `public` e `private`. A fim de garantir o acesso da subclasse a métodos protegidos da superclasse, estes precisam ser

implementados como públicos, tornando-os acessíveis também a qualquer outra classe. Já os atributos protegidos precisam ser especificados como privados, a fim de não liberar também o acesso aos atributos privados da classe. Com isso a subclasse é forçada a recorrer a métodos de acesso para manipular os atributos protegidos.

Pudemos também identificar uma dificuldade com relação ao mecanismo de emulação de herança, o qual aparentemente passou despercebido aos outros autores. Consideremos por exemplo a definição de **ClasseAbstrata** na Figura 7.1 . Uma classe abstrata pode ser implementada de maneira análoga a uma classe concreta, apenas definindo seus métodos abstratos como privados e não fornecendo nenhum construtor ou destrutor. A implementação de `metodo1()` inclui uma chamada a `metodo2()`:

```
subroutine ClasseAbstrata_metodo1(this)
type(ClasseAbstrata), intent(inout) :: this

print *, metodo2(this)

end subroutine ClasseAbstrata_metodo1

! método abstrato:
function ClasseAbstrata_metodo2(this) result(numero)
type(ClasseAbstrata), intent(in) :: this
real :: numero

numero = 0.0

end function ClasseAbstrata_metodo2
```

Consideremos agora a **ClasseConcreta**, a qual herda o método concreto de **ClasseAbstrata** e sobrepõe o seu método abstrato (note a chamada de método para acessar o atributo protegido):

```
subroutine ClasseConcreta_metodo1(this)
type(ClasseConcreta), intent(inout) :: this

call metodo1(this%super)

end subroutine ClasseConcreta_metodo1

function ClasseConcreta_metodo2(this) result(numero)
type(ClasseConcreta), intent(in) :: this
real :: numero

numero = getAtributo(this) * getAtributo(this)

end function ClasseConcreta_metodo2
```

Surpreendentemente, ao invocarmos `metodo1()` em uma instância de **ClasseConcreta** a implementação de `metodo2()` executada será aquela especificada na **ClasseAbstrata** ao invés do

método sobreposto! Isto porque a implementação de `metodo1()` na classe abstrata foi ligada à implementação de `metodo2()` em tempo de compilação, com isso levando a este comportamento incorreto. O segundo programa de exemplo no apêndice C ilustra a diferença obtida comparando-se as saídas dos códigos em Fortran e Java.

Para corrigir este problema não há outra solução senão repetir a implementação de `metodo1()` da **`ClasseAbstrata`** na **`ClasseConcreta`**, com isso gerando duplicação de código e dificultando a manutenção de nossa aplicação.

O exemplo que criamos está longe de representar uma situação incomum de ser encontrada na prática, pois este problema chega a se manifestar até mesmo na implementação de alguns padrões de projeto, como o padrão *Template Method* (Gamma et al. 1995). Este padrão serve para implementar o esqueleto de um algoritmo através de um `metodoGabarito()`, que chama métodos que implementam os passos individuais do algoritmo. Uma subclasse pode então especializar alguns passos do algoritmo sobrepondo alguns destes métodos, como mostra a Figura 7.3 .

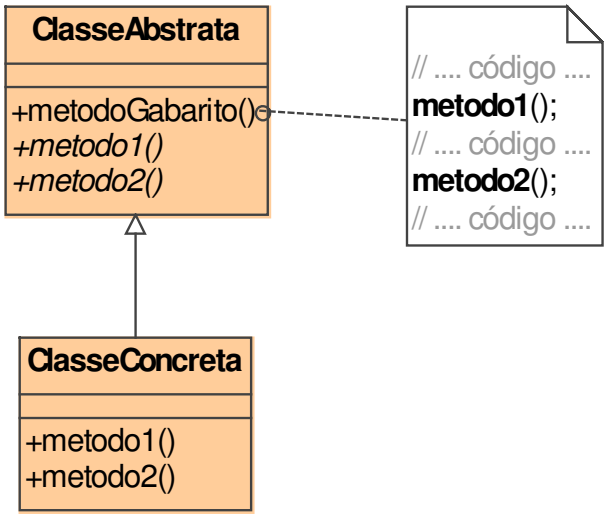


Figura 7.3: O padrão *Template Method*.
Adaptado de Gamma et al. (1995).

7.2.3. Polimorfismo

Para emular o conceito de polimorfismo é preciso criar um novo tipo de dados contendo ponteiros para cada uma das classes que serão manipuladas polimorficamente. Considerando por exemplo a interface **`Funcao`** na Figura 7.1, teremos que, em tempo de execução, ela poderá se referir a uma instância de **`Ancestral`** ou **`Descendente`**. Assim, uma possível implementação seria:

```
type, public :: Funcao
    ! ponteiros para classes que implementam a interface:
    private
    type(Ancestral),   pointer :: ancestral   => null()
    type(Descendente), pointer :: descendente => null()
end type Funcao
```

, sendo que em tempo de execução apenas um dos ponteiros estará associado.

A implementação do método polimórfico `avalie()` simplesmente testa qual dos ponteiros está associado e então delega a tarefa à classe apropriada:

```
function Funcao_avalie (this,x) result(valor)
type(Funcao), intent(in) :: this
real, intent(in) :: x
real :: valor

! delega a tarefa ao objeto apropriado:
if (associated(this%ancestral)) then
    valor = avalie(this%ancestral,x)
else if (associated(this%descendente)) then
    valor = avalie(this%descendente,x)
else
    write (6,*) "Erro! Funcao_avalie: nenhum objeto alocado!"
    stop
end if

end function Funcao_avalie
```

Também é necessário criar um construtor para cada classe polimórfica e então sobrecarregá-los no construtor da interface:

```
interface new
    module procedure Funcao_newAncestral
    module procedure Funcao_newDescendente
end interface new
```

Podemos agora então manipular instâncias de **`Ancestral`** ou **`Descendente`** através da interface **`Funcao`**:

```
type(Funcao) :: f
integer :: numero

call new(f,"Ancestral",1.0)
numero = avalie(f,2.0)
call finish(f)

call new(f,"Descendente",3.0,4.0)
numero = avalie(f,2.0)
call finish(f)
```

, o que seria equivalente a, em Java:

```
int numero;
Funcao f = new Ancestral(1.0F);
numero = f.avalie(2.0F);
f = new Descendente(3.0F,4.0F);
numero = f.avalie(2.0F);
```

Outra alternativa seria criar construtores que associem os ponteiros a instâncias já existentes. Neste caso teríamos então:

```
type(Ancestral),   target :: meuAncestral
type(Descendente), target :: meuDescendente
type(Funcao) :: f
integer :: numero

call new(meuAncestral,1.0)
call new(f,meuAncestral)
numero = avalie(f,2.0)
call finish(f)

call new(meuDescendente,3.0,4.0)
call new(f,meuDescendente)
numero = avalie(f,2.0)
call finish(f)
```

, o que seria equivalente a, em Java:

```
int numero;
Ancestral meuAncestral = new Ancestral(1.0F);
Funcao f = meuAncestral;
numero = f.avalie(2.0F);
Descendente meuDescendente = new Descendente(3.0F,4.0F);
f = meuDescendente;
numero = f.avalie(2.0F);
```

Para manipular instâncias de subclasses polimorficamente através da superclasse o procedimento é análogo: basta criar um novo tipo contendo ponteiros para todas as possíveis subclasses. A única diferença em relação à implementação de uma interface é que seria necessário implementar neste novo tipo todos os métodos da superclasse, e não apenas os métodos da interface como fizemos em nosso exemplo.

Em nossa convenção o novo tipo possui o nome da superclasse antecedido de `Poly`. Assim, para manipular polimorficamente as classes **`Ancestral`** e **`Descendente`** teríamos o tipo denominado **`PolyAncestral`**.

Concluímos então que o procedimento para emular o conceito de polimorfismo em Fortran 90/95 consiste em criar um novo tipo que agrega uma instância de qualquer um dos tipos polimórficos, e cujos métodos polimórficos simplesmente delegam o trabalho às classes apropriadas, como mostra a Figura 7.2 .

7.2.4. Outras funcionalidades

A implementação de códigos razoavelmente elaborados em Fortran utilizando POO requer ainda a utilização de algumas funcionalidades extras que não foram consideradas pelos outros autores, as quais reunimos nesta seção.

Em aplicações muito grandes é preciso haver um mecanismo para desfazer conflitos de nomes entre classes homônimas de diferentes bibliotecas. Suponhamos que organizamos as classes da Figura 7.1 no pacote `meuPacote`. Em nossa convenção de estilo, o módulo contendo a classe **`Ancestral`** será denominado `meuPacote_Ancestral_Class`. Para importá-lo, outras unidades deverão incluir o comando:

```
use meuPacote_Ancestral_Class
```

Caso a classe **`Ancestral`** entre em conflito com uma classe homônima em um outro pacote, poderemos desfazer a ambigüidade da seguinte maneira:

```
use meuPacote_Ancestral_Class
use outroPacote_Ancestral_Class, outroPacote_Ancestral => Ancestral
```

, e bastará então nos referirmos à segunda classe pelo seu nome completo `outroPacote_Ancestral`. Note que se a classe a ser renomeada possui atributos ou métodos estáticos, todos eles também necessitarão ser renomeados.

Por conveniência, podemos ainda reunir em um único módulo todos os outros módulos das classes do pacote:

```
! Pacote com todas as classes e interfaces.
module meuPacote_Package

! classes públicas continadas no pacote:
use meuPacote_Ancestral_Class
use meuPacote_Descendente_Class
use meuPacote_Funcao_Interface
use meuPacote_ClasseAbstrata_Class
use meuPacote_ClasseConcreta_Class

! acessibilidade padrão:
public

end module meuPacote_Package
```

, e este será o único módulo a ser importado caso se deseje obter acesso a todas as classes do pacote. Teremos então que os pacotes em Fortran são implementados como mostra a Figura 7.2 .

Outra funcionalidade que se faz necessária em aplicações razoavelmente complexas é a capacidade de determinar em tempo de execução a classe exata de um tipo polimórfico. Para tanto,

basta incluir um método `getClassName()` no tipo polimórfico. Considerando a interface **`Funcao`** vista anteriormente teríamos:

```
function Funcao_getClassName(this) result(classname)
type(Funcao), intent(in) :: this
character(len=100) :: classname

if ( associated(this%ancestral) ) then
    classname = "meuPacote_Ancestral"
else if ( associated(this%descendente) ) then
    classname = "meuPacote_Descendente"
else
    classname = "null"
end if

end function Funcao_getClassName
```

Muitas vezes desejamos ainda voltar a manipular um tipo polimórfico através da classe mais específica, ao invés da classe mais genérica. Esta funcionalidade também pode ser facilmente implementada adicionando um método `toClasse()`, que retornaria um ponteiro para a componente `this%classe` do tipo polimórfico:

```
function Funcao_toDescendente(this) result(f)
type(Funcao), intent(in) :: this
type(Descendente), pointer :: f

if ( associated(this%descendente) ) then
    f => this%descendente
else
    write (6,*) "Erro! meuPacote:Funcao:toDescendente: este objeto&
                & nao eh uma instancia de Descedente"
    stop
end if

end function Funcao_toDescendente
```

Assim, se em tempo de execução temos um objeto de **`Descedente`** sendo referenciado através da interface **`Funcao`**, podemos voltar a manipulá-lo como uma instância de **`Descendente`** do seguinte modo:

```
type(Funcao) :: f
type(Descendente), pointer :: meuDescendente

call new(f,"Descendente",3.0,4.0)

! .... código ....

if ( getClassName(f) == "meuPacote_Descendente" ) then
    meuDescendente => toDescendente(f)
    call setAtributo2(meuDescendente,5.0)
end if
```

Uma outra funcionalidade eventualmente necessária em algumas aplicações é a de persistência, para que um dado objeto possa armazenar o seu estado em memória permanente para utilização posterior.

Para possibilitar que uma classe armazene os seus dados, é preciso dotá-la de métodos `readObject()` e `writeObject()`, e também de um construtor sem parâmetros. O método `writeObject()` primeiro escreve o nome da classe, e a seguir os dados necessários para recuperar o estado do objeto. Para recriar um objeto a partir da memória permanente, primeiramente chamamos o seu construtor sem parâmetros e então invocamos o método `readObject()`. Este método lê diretamente os dados referentes ao objeto, pois a leitura do nome da classe fica a cargo de quem enviou a mensagem `readObject()`.

Com estas regras uma determinada classe também pode ser lida ou escrita polimorficamente. A implementação do método `readObject()` do tipo polimórfico primeiramente lê o nome da classe para então alocar a componente correta e invocar o método `readObject()` correspondente. O método `writeObject()` do tipo polimórfico escreve o nome da classe polimórfica e então delega a tarefa para o método `writeObject()` do objeto alocado em tempo de execução.

Uma restrição particularmente embaraçosa à implementação de projetos orientados a objetos em Fortran, e que passou despercebida pelos outros autores, reside no fato de não ser possível dois módulos se acessarem mutuamente. Assim, o código abaixo não pode ser compilado:

```
module moduloA

use moduloB  ! ilegal!

end module moduloA

module moduloB

use moduloA  ! ilegal!

end module moduloB
```

Com isso associações bidirecionais, como a ligando a classe **A** à classe **B** na Figura 7.4, seriam difíceis de serem implementadas.

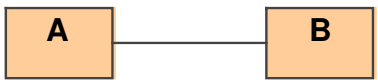


Figura 7.4: Uma associação bidirecional.

Também não é possível a um módulo acessar a si mesmo indiretamente através de outros módulos:

```
module moduloA

use moduloB  ! ilegal!

end module moduloA

module moduloB

use moduloC  ! ilegal!

end module moduloB

module moduloC

use moduloA  ! ilegal!

end module moduloC
```

Assim sendo, também esbarraríamos na mesma dificuldade ao tentar implementar em Fortran o projeto da Figura 7.5 . Esta é uma forma mais sutil da restrição acima, pois ao projetar um sistema complexo é preciso se certificar ainda que não exista uma única rota que possa ser circularmente navegada no diagrama de classes.

Neste ponto também é preciso estar atento às regras de emulação dos conceitos de POO em Fortran. Por exemplo, o padrão *Composite* visto anteriormente na Figura 6.11 na verdade seria implementado em Fortran pelo diagrama da Figura 7.6, no qual as classes **`PolyCélulaAbstrata`** e **`CélulaComposta`** se acessam mutuamente.

A dificuldade com módulos que se acessam recursivamente pode ser contornada incluindo várias classes em um mesmo módulo, desde que as componentes recursivas sejam declaradas como ponteiros. Por exemplo, para implementar a associação bidirecional entre as classes da Figura 7.4 teríamos:

```
type, public :: A
    type(B), pointer :: atributo
end type A

type, public :: B
    type(A), pointer :: atributo
end type B
```

Porém esta alternativa acarretaria uma penalidade muito alta por abrir mão completamente do encapsulamento: a classe **`A`** teria acesso irrestrito a todos os atributos e métodos da classe `B`, e vice-versa. Esta seria uma alternativa ainda mais inviável se o caminho circular abranger um número considerável de classes.

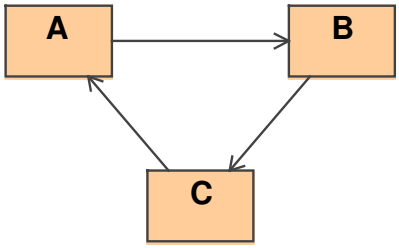


Figura 7.5: Um projeto que envolveria módulos recursivos em Fortran.

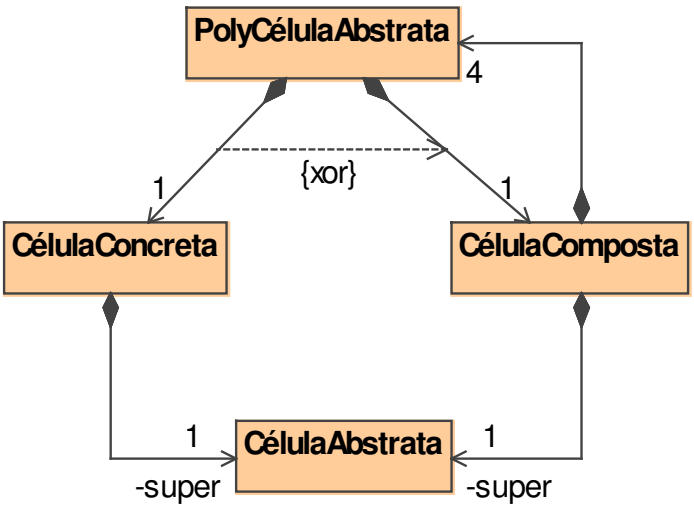


Figura 7.6: Implementação do padrão *Composite* em Fortran.

Note que é possível a uma classe abrigar uma instância de si mesma, desde que esta também seja declarada como um ponteiro:

```
type, public :: Classe
    type(Classe), pointer :: atributo
end type Classe
```

Por fim, encerramos a nossa discussão acerca da implementação de funcionalidades avançadas de POO em Fortran apresentando a implementação do padrão *Singleton* na Listagem 7.2, tal como foi apresentada a implementação em Java na Listagem 6.4 . A seqüência para acessar a instância *Singleton* seria:

```
type(Singleton), pointer :: instancia

instancia => Singleton_getInstance()
! .... código ....
call Singleton_finishInstance()
```

```
! O padrão Singleton.
module Singleton_Class

implicit none

private

type, public :: Singleton
    private
    real :: attribute
end type Singleton

type(Singleton), pointer :: singletonInstance => null()

save

public :: Singleton_getInstance, Singleton_finishInstance

interface new
    module procedure Singleton_new
end interface new

interface finish
    module procedure Singleton_finish
end interface finish

contains

!********************
!* métodos públicos *
!********************

function Singleton_getInstance() result(instance)
type(Singleton), pointer :: instance

if ( .not. associated(singletonInstance) ) then
    allocate(singletonInstance)
    call new(singletonInstance)
end if

instance => singletonInstance

end function Singleton_getInstance

subroutine Singleton_finishInstance()

call finish(singletonInstance)
deallocate(singletonInstance)

end subroutine Singleton_finishInstance

!********************
!* métodos privados *
!********************
```

```
! Construtor:
subroutine Singleton_new(this)
type(Singleton), intent(inout) :: this

! .... código ....

end subroutine Singleton_new

! Destrutor:
subroutine Singleton_finish(this)
type(Singleton), intent(inout) :: this

! .... código ....

end subroutine Singleton_finish

end module Singleton_Class
```

Listagem 7.2: O padrão *Singleton* em Fortran.

7.3. Uma olhada no Fortran 2003

O novo padrão da linguagem, conhecido informalmente como Fortran 2003, trará inúmeras facilidades a fim de tornar mais prática a implementação da POO. São essas novas características, das quais podemos ressaltar o suporte direto a herança e polimorfismo, que passamos a apresentar em detalhes nas seções seguintes.

Nossa discussão baseia-se principalmente nos artigos de Cohen (1999) e Reid (2002), além da versão atual do rascunho do novo padrão atualmente disponível na WWW nas páginas dos comitês responsáveis (J3; WG5).[F]

Chamamos a atenção do leitor para o fato de a versão atual do rascunho estar sujeita a mudanças até o final do processo de votação, o qual deverá ocorrer por volta de 12/2004. Porém, no estágio atual do processo de padronização, o rascunho do padrão poderá receber apenas pequenas alterações, do contrário a adoção do novo padrão será prorrogada por pelo menos mais um ano.

Nas seções 7.3.1 e 7.3.2 introduzimos a nova sintaxe do Fortran 2003. Devido ao fato de ainda não existirem compiladores disponíveis para testar os códigos de exemplo, o leitor deverá contar apenas com a nossa interpretação do rascunho do padrão em alguns casos. Entretanto acreditamos que a discussão apresentada aqui nos possibilitará ter uma idéia bem clara a respeito do que podemos esperar da novíssima versão do padrão do Fortran. Por fim, algumas críticas são tecidas na seção 7.3.3 .

[F] Na data em que este trabalho foi redigido (12/2003) o número de referência do documento disponível era ISO/IEC JTC1/SC22/WG5 N1578, datado de 08/10/2003.

7.3.1. Classes

Em Fortran 2003 será possível associar rotinas diretamente aos novos tipos de dados. Considerando a classe **Ancestral** em nosso exemplo (Figura 7.1), teríamos a seguinte implementação:

```
type, public :: Ancestral
    private
    real, public :: atributo1
    contains
    procedure, public :: setAtributo1 => Ancestral_setAtributo1
    ! .... código ....
end type Ancestral
```

, ou seja, estamos associando a rotina `Ancestral_setAtributo1()` diretamente ao tipo **Ancestral** através do nome `setAtributo1`. Com isto não é mais necessário incluir o bloco de interface `setAtributo1`, como fizemos na Listagem 7.1 . Outra melhoria é que agora as visibilidades dos atributos (e métodos) podem ser fixadas individualmente. Assim como em Fortran 90/95, as visibilidades não especificadas são assumidas como `public`, sendo que este comportamento padrão pode ser alterado incluindo-se a palavra-chave `private` no início da lista de componentes.

A rotina `Ancestral_setAtributo1()` será implementada praticamente da mesma maneira que em Fortran 90/95 (Listagem 7.1):

```
subroutine Ancestral_setAtributo1 (this,valor1)
class(Ancestral), intent(inout) :: this
real, intent(in) :: valor1

this%atributo1 = valor1

end subroutine Ancestral_setAtributo1
```

As novas regras de sintaxe permitem invocar o método `setAtributo1()` através da notação:

```
call meuAncestral%setAtributo1(1.0)
```

A declaração do método `setAtributo1()` acima implica que a rotina `Ancestral_setAtributo1()` deve obrigatoriamente aceitar uma instância de **Ancestral** em primeiro lugar na sua lista de parâmetros. Caso não desejemos que a instância na qual o método foi invocado seja passada para a rotina, como é o caso da implementação de métodos estáticos, devemos então utilizar a palavra-chave `nopass`:

```
procedure, public , nopass :: getNInstancias => Ancestral_getNInstancias
```

, com isto o método estático fica disponível também através da instância.

O comportamento padrão pode ser explicitamente confirmado:

```
procedure, public , pass :: setAtributo1 => Ancestral_setAtributo1
```

, ou então podemos associar a instância a um outro argumento que não necessariamente o primeiro da lista:

```
procedure, public , pass(this) :: setAtributo1 => Ancestral_setAtributo1
```

É possível sobrecarregar um método com mais de uma rotina utilizando a palavra-chave `generic` ao invés de `procedure`:

```
generic, public, pass :: metodo => metodo1, metodo2
```

Construtores poderão ser implementados diretamente sobrecarregando o nome do novo tipo com uma função, a qual irá sobrepor ou sobrecarregar o construtor de estrutura padrão:

```
interface Ancestral
    module procedure construtor
end interface Ancestral

! .... código ....

function construtor(valor1) result(instancia)
type(Ancestral)  :: instancia
real, intent(in) :: valor1

call instancia%setAtributo1(valor1)
call Ancestral_incNInstancias()

end function construtor
```

Com isso é possível inicializar as variáveis de um tipo mesmo que ele contenha componentes privadas. Em nosso exemplo, o construtor acima sobrepõe o construtor de estrutura padrão de Ancestral:

```
meuAncestral = Ancestral(1.0)
```

Destrutores são implementados ligando-se a rotina ao tipo através da palavra-chave `final`, ao invés de `procedure` ou `generic`:

```
final :: Ancestral_finish
```

, e tais rotinas serão então automaticamente chamadas sempre que uma determinada instância deixar de existir. As componentes do tipo que também são tipos derivados são finalizadas logo em seguida, e assim sucessivamente.

Um determinado método pode ter a sua implementação protelada, a fim de implementar classes abstratas:

```
type, public, abstract :: ClasseAbstrata
    real :: atributo
    contains
    ! .... código ....
    procedure(ClasseAbstrata_metodo2), deferred, pass :: metodo2
end type ClasseAbstrata
```

, e a forma geral do método abstrato é especificada da seguinte maneira:

```
abstract interface
    real function ClasseAbstrata_metodo2(this)
    class(ClasseAbstrata), intent(in) :: this
    end function ClasseAbstrata_metodo2
end interface
```

Interfaces podem ser implementadas como classes abstratas puras.

7.3.2. Herança e polimorfismo

A herança será implementada em Fortran 2003 seguindo a sintaxe:

```
type, public, extends(Ancestral) :: Descendente
    real :: atributo2
    contains
    procedure :: getAtributo2 => Descendente_getAtributo2
    procedure :: setAtributo2 => Descendente_setAtributo2
    procedure :: avalie => Descendente_avalie
end type Descendente
```

O novo tipo automaticamente herda todos os atributos e métodos da classe ancestral. Assim, o atributo `atributo1` de **`Ancestral`** (Figura 7.1) poderá ser acessado diretamente como `meuDescendente%atributo1`. O tipo ancestral também estará automaticamente disponível como um todo através de uma componente com o mesmo nome no tipo descendente. Em nosso exemplo esta componente seria `meuDescendente%Ancestral`, e os atributos da classe ancestral poderão alternativamente serem acessados como `meuDescendente%Ancestral%atributo1`.

No exemplo acima, sobrepomos o método `avalie()` na classe descendente. Caso desejássemos impedir a sobreposição bastaria declarar este método como `non_overridable` em sua definição na classe ancestral:

```
procedure, public , non_overridable :: avalie => Ancestral_avalie
```

Uma variável polimórfica pode ser declarada com a palavra-chave `class` no lugar de `type`, e poderá então referenciar também qualquer subclasse do tipo declarado. Em nosso exemplo,

supondo que a classe **Ancestral** estende a interface **Funcao** (implementada como uma classe abstrata pura) teríamos:

```
integer :: numero
type(Ancestral),   target :: meuAncestral
type(Descendente), target :: meuDescendente
class(Funcao), pointer :: f

meuAncestral   = Ancestral(1.0)
meuDescendente = Descendente(3.0,4.0)
f => meuAncestral
numero = f%avalie(2.0)
f => meuDescendente
numero = f%avalie(2.0)
```

, ou, equivalentemente:

```
integer :: numero
class(Funcao), pointer :: f

allocate ( f, source = Ancestral(1.0) )
numero = f%avalie(2.0)
deallocate (f)

allocate ( f, source = Descendente(3.0,4.0) )
numero = f%avalie(2.0)
deallocate(f)
```

, sendo que o parâmetro `source` inicializa `f` com uma instância com o mesmo conteúdo e do mesmo tipo dada pela expressão à direita da igualdade, em nosso caso os construtores das classes **Ancestral** e **Descendente** (que sobrepõem os respectivos construtores de estrutura padrão).

Uma variável polimórfica também poderá ser declarada como `class(*)`, a qual poderá então se referir a qualquer classe sem restrições.

Para inferir o tipo exato de uma variável polimórfica em tempo de execução temos as funções intrínsecas `same_type_as(a,b)`, que testa se `a` e `b` são do mesmo tipo, e `extends_type_of(a,b)`, que testa se a classe de `a` é uma subclasse de `b`.

O comando `select type` executa no máximo um dos seus blocos constituintes, dependendo do tipo de uma variável em tempo de execução:

```
f => meuAncestral
select type ( ff => f )
type is (Descendente)
    print *, ff%getAtributo2()
class is (Funcao)
    print *, ff%avalie(2.0)  ! este bloco é executado
end select
```

Em cada um dos blocos `ff` assume o mesmo tipo ou classe declarados nos comandos `type is` ou `class is`. As regras para a escolha do bloco a ser executado são:

- Se algum bloco `type is` se ajusta, ele é tomado;
- caso contrário, se um único bloco `class is` se ajusta, ele é tomado;
- caso contrário, se vários blocos `class is` se ajustam então um deles deve ser uma extensão de todos os outros, e é tomado.

Também pode haver um bloco `class default`, o qual é escolhido caso nenhum outro bloco seja selecionado. Neste bloco `ff` tem o mesmo tipo de `f`.

Encerramos aqui a nossa discussão acerca da sintaxe proposta no Fortran 2003 para incorporar herança e polimorfismo. Com o suporte direto a estes conceitos, o problema relacionado à implementação do padrão *Template Method* apontado na seção 7.2.2 não mais se apresenta, e também não é mais necessário criar os tipos polimórficos como fizemos na seção 7.2.3 .

7.3.3. Algumas omissões

As novidades introduzidas no Fortran 2003 facilitarão em muito a implementação de códigos orientados a objetos. Entretanto, em nossa opinião o padrão atualmente em processo de votação não é plenamente satisfatório por deixar de fora algumas facilidades básicas presentes em outras linguagens.

Em primeiro lugar, a sintaxe proposta para incorporar métodos aos novos tipos (seção 7.3.1), com as chamadas *rotinas ligadas ao tipo*[F], é uma espécie de "implementação de baixo-nível" de classes. Nela, o programador é obrigado a cuidar de todos os detalhes os quais em outras linguagens são lidados automaticamente pelo compilador de forma transparente. De fato, surpreendentemente não há muita diferença entre a nova sintaxe e a maneira empregada na emulação de classes em Fortran 90/95 (seção 7.2.1). Nas outras linguagens não é preciso declarar explicitamente uma instância da classe na lista de parâmetros dos métodos, e dentro do escopo da classe também não é necessário se referir aos atributos da instância precedendo-os pelo seu nome (em nossa convenção, `this%...`). Considerando a classe `Ancestral` em nosso exemplo, uma possível implementação mais satisfatória de classes seria:

[F] Em inglês, *type-bound procedures*.

```
type, public :: Ancestral

    real, public :: atributo1

    contains

    ! .... código ....

    subroutine, public :: setAtributo1(valor1)
    real, intent(in) :: valor1

    atributo1 = valor1

    end subroutine setAtributo1

    ! .... código ....

end type Ancestral
```

Uma omissão no novo padrão são os atributos e métodos estáticos. Estes terão que continuar sendo declarados e acessados com uma notação inconsistente como em Fortran 90/95: `Ancestral_nInstancias` e `Ancestral_getNInstancias()`, ao invés de `Ancestral%nInstancias` e `Ancestral%getNInstancias()`. Também não há nada que force o programador a adotar uma notação consistente na escolha dos nomes acima. A notação fica ainda mais inconsistente quando consideramos que na subclasse os atributos e métodos estáticos não estarão automaticamente disponíveis como `Descendente%nInstancias` e `Descendente%getNInstancias()`.

Ainda com relação à herança, outra omissão é a inexistência do especificador de acesso protegido. No Fortran 2003 será introduzida a palavra-chave `protected`, porém com um significado completamente diferente: indicar que uma variável global não pode ser alterada fora do escopo do módulo onde se encontra. Assim, com a declaração:

```
real, protected :: a
```

, o valor da variável `a` pode ser acessado externamente, mas não pode ser modificado.

Da mesma maneira que em Fortran 90/95, sem o especificador de acesso protegido, métodos protegidos terão que ser especificados como públicos para serem acessíveis à subclasse. Atributos protegidos terão que ser declarados como públicos ou privados, neste último caso forçando a subclasse a recorrer a métodos de acesso.

Falta portanto um conjunto mais rico de especificadores de acesso em Fortran 2003. Em Java, por exemplo, o especificador `package` é muito útil na construção de bibliotecas de classes, pois esconde do usuário detalhes que só dizem respeito à implementação da biblioteca e só devem ser motivo de preocupação apenas do programador que a criou.

Outra omissão com relação à herança diz respeito à herança múltipla. Fortran 2003 implementa apenas a herança simples, sendo que às vezes é necessário a uma classe-filha herdar as características de mais de uma classe-pai ao mesmo tempo. A experiência com outras linguagens mostra que isso é útil na construção de novas interfaces, ou ainda quando uma classe necessita implementar várias interfaces.

O inconveniente apresentado na seção 7.2.4 relacionado a módulos que se acessam recursivamente continua presente em Fortran 2003, de modo que certos projetos orientados a objetos poderão ser mapeados para código Fortran apenas às custas do sacrifício do encapsulamento.

7.4. Conclusões

Fortran sempre foi tradicionalmente a linguagem mais adotada em aplicações científicas e de engenharia. Sua ênfase é em procurar obter o máximo de desempenho possível possibilitando gerar código que possa ser executado eficientemente pela máquina, e não em criar e manter programas complexos e livres de erros (Java estaria no outro extremo do espectro). Porém, com o crescimento da escala de complexidade de nossos códigos, acreditamos que esta não seja mais a direção apropriada a seguir, afinal de contas obter a resposta correta sempre será mais importante do que chegar até ela rapidamente!

O paradigma da POO facilita enormemente desenvolver e manter aplicações complexas, porém este modelo de programação não é diretamente suportado pelo padrão atual do Fortran. Embora seja factível realizar POO em Fortran 90/95, o processo de emulação dos conceitos de herança e polimorfismo é extremamente trabalhoso e pode até mesmo ser inviável em certos problemas, em particular aqueles que implementam uma hierarquia de herança muito longa.

Seguramente podemos experimentar alguns dos benefícios da POO seguindo este modelo, se estamos dispostos a programar de uma maneira disciplinada. Entretanto em algumas aplicações este procedimento só fará sentido se encarado como um investimento no Fortran 2003.

Infelizmente a abordagem adotada na elaboração do novo padrão do Fortran é extremamente minimalista, e não o colocará em pé de igualdade nem mesmo com outras linguagens orientadas a objetos já existentes há mais de uma década. Certas funcionalidades existentes em algumas linguagens - tais como *templates* - estão ausentes, e mesmo algumas características básicas foram deixadas de fora, como atributos e métodos estáticos e o especificador de acesso protegido.

O suporte a POO foi um tema polêmico durante a elaboração do padrão do Fortran 90, e que foi deixado de lado para a próxima revisão geral da linguagem. Na verdade o Fortran poderia até mesmo se beneficiar em implementar tardiamente a POO, aprendendo com os erros e acertos das outras linguagens e incorporar certas funcionalidades avançadas tais como asserções. Essa

vantagem entretanto não foi apreciada pelos comitês responsáveis pela padronização do Fortran, pelo que os fãs de POO terão que aguardar outros 10 anos até a próxima revisão geral.

Futuramente aqueles que tiverem que manter códigos extremamente complexos, e não estiverem dispostos a abrir mão da eficiência e da expressividade do Fortran em lidar com códigos numéricos, provavelmente terão que aprender mais do que uma única linguagem de programação.

No momento é difícil conciliar o Fortran com outras linguagens. O mais simples seria combinar código compilado do Fortran com código compilado do C++, porém isto em geral é possível apenas nos casos em que ambos os compiladores são do mesmo fabricante. Assim, também não é possível unir as duas linguagens de maneira portátil, pois o código teria que ser alterado sempre que fosse transportado para uma plataforma diferente. A interoperabilidade com o ANSI C é um dos requisitos principais do Fortran 2003, o qual poderá então ser utilizado como uma ponte para o C++.

Considerando todos os fatores acima, optamos por implementar o banco de dados atômicos em Fortran, pois o seu projeto em software (o qual apresentaremos no capítulo seguinte) não é muito susceptível às deficiências mais graves do Fortran 90/95, e preferimos não nos preocupar com questões relacionadas à interoperabilidade com o C++ e à portabilidade.

Referências

Akin, J. E., & Singh, M. 2002, Advances in Engineering Software, 33, 461

Cary, J. R., Shasharina, S. G., Cummings, J. C., Reynders, J. V. W., & Hinker, P. J. 1997, Comp. Phys. Comm., 105, 20

Clerman, N. S. 1999, ACM SIGPLAN Fortran Forum, 18(3), 29

Cohen, M. 1999, ACM SIGPLAN Fortran Forum, 18(1), 14

Decyk, V. K., & Norton, C. D. 1999, ACM SIGPLAN Fortran Forum, 18(3), 26

---. 2000a, ACM SIGPLAN Fortran Forum, 19(1), 19

---. 2000b, ACM SIGPLAN Fortran Forum, 19(3), 8

Decyk, V. K., Norton, C. D., & Szymanski, B. K. 1996, *"Introduction to Object-Oriented Concepts using Fortran90"*, Technical Report PPG-1560 (Institute of Plasma and Fusion Research, UCLA Dept. of Physics and Astronomy)

---. 1997a, ACM SIGPLAN Fortran Forum, 16(1), 13

---. 1997b, Scientific Programming, 6, 363

---. 1998, Comp. Phys. Comm., 115, 9

---. 1999, ACM SIGPLAN Fortran Forum, 18(2), 22

Dupée, B. J. 1994, ACM SIGPLAN Fortran Forum, 13(1), 21

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. 1995, *"Design Patterns"* (Addison-Wesley)

Gray, M. G., & Roberts, R. M. 1997, Computers in Physics, 11, 355

J3, Comitê J3 de padronização do Fortran: http://www.j3-fortran.org/

Kim, Y.-H., Lee, I.-H., & Martin, R. M. 2000, Comp. Phys. Comm., 131, 10

Machiels, L., & Deville, M. O. 1997, ACM Trans. Math. Software, 23(1), 32

Metcalf, M., & Reid, J. 1999, *"Fortran 90/95 explained"*, 2ª ed. (Oxford: Oxford University Press)

Norton, C. D. 1996, tese de doutorado, *"Object-Oriented Programming Paradigms in Scientific Computing"*, (Rensselaer Polytechnic Institute, Troy, NY)

Norton, C. D., Decyk, V. K., & Szymanski, B. K. 1996, ACM SIGAPP Applied Computing Review, 4(1), 27

Norton, C. D., Szymanski, B. K., & Decyk, V. K. 1995, Communications of the ACM, 38(10), 88

Qiang, J., Ryne, R. D., & Habib, S. 2000, Comp. Phys. Comm., 133, 18

Reid, J. 2002, ACM SIGPLAN Fortran Forum, 21(2), 1 (Versões atualizadas deste artigo disponíveis na página do comitê WG5: http://www.nag.co.uk/sc22wg5/ .)

WG5, Comitê WG5 de padronização do Fortran: http://www.nag.co.uk/sc22wg5/

8. A Implementação em Software do Banco de Dados Atômicos

8.1. O projeto do banco de dados atômicos

Utilizando a abordagem da programação orientada a objetos, podemos agora mapear os conceitos do BDA diretamente para as componentes da sua implementação em *software*.

A estrutura desenvolvida para os arquivos de dados atômicos, discutida no capítulo 3, permite a inclusão das taxas como tabelas, ajustes ou ainda como uma combinação de tabelas e ajustes. Esta estrutura é modelada pelo diagrama de classes da Figura 8.1, resultando nas classes **`Table`**, **`Fit`** e **`Composite`**[F]. Estas três classes especializam a classe abstrata **`UnivariateF`**, que representa uma função de uma variável qualquer, cujo valor é retornado pelo método `eval()`. A classe **`Table`** implementa este método interpolando linearmente sobre uma tabela de pontos, armazenada nos seus atributos `x[]` e `y[]`.

Já a classe **`Fit`** implementa o método `eval()` através de um ajuste analítico, cujos parâmetros estão armazenados no atributo `c[]`. Na verdade este método é abstrato e é implementado apenas pelas inúmeras subclasses **`Fitn`**, cada qual implementando especificamente uma das fórmulas matemáticas reunidas no apêndice A. O usuário pode então estender o BDA com novos ajustes analíticos simplesmente criando uma nova subclasse da classe **`Fit`**, e implementando a fórmula matemática correspondente no método `eval()`.

A classe **`Composite`** implementa uma composição de funções de uma dimensão individuais, as quais podem ser tabelas, ajustes ou até mesmo outras composições. O seu método `eval()` simplesmente chama os métodos correspondentes das componentes individuais, e então adiciona os valores retornados. Temos aqui portanto mais um exemplo da aplicação do padrão *Composite* (Gamma et al. 1995), o qual não pode ser implementado adequadamente em Fortran, como vimos no capítulo anterior. Felizmente, em nosso problema em particular a recursão não é imprescindível, e podemos contornar esta dificuldade introduzindo uma classe adicional IF (de *implementation*

[F] Para que o BDA possa ser utilizado e estendido por outros autores, todo o seu projeto e implementação foram feitos em inglês.

function), como mostra a Figura 8.2 . A única penalidade resultante é a de complicarmos o nosso projeto com a inclusão de uma classe a mais que não desempenha nenhum papel.

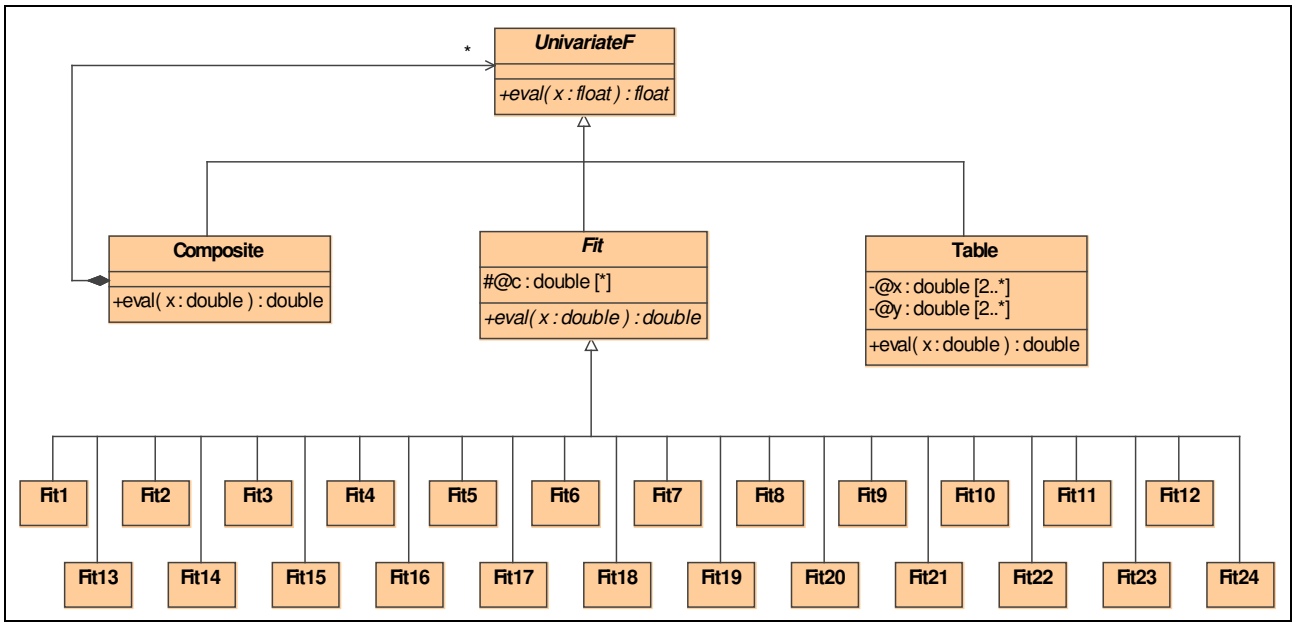


Figura 8.1: Classes básicas do BDA.

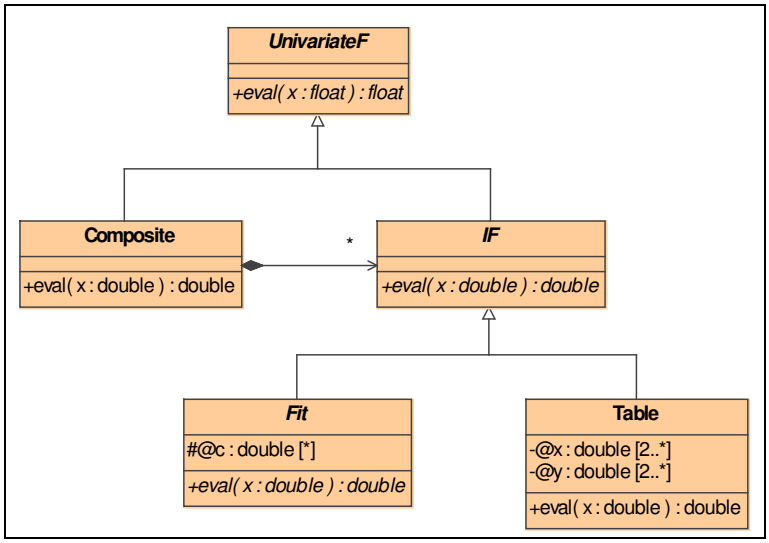


Figura 8.2: Improviso para implementar as classes básicas do BDA em Fortran.

A implementação do BDA em *software* usando programação orientada a objetos permite que o usuário modifique as formas pelas quais são especificadas as taxas nos arquivos de dados atômicos sem afetar o código. Isto porque internamente todas as taxas são manipuladas polimorficamente como funções de uma variável, quer sejam elas implementadas como tabelas, ajustes ou composições. A dificuldade apontada na seção 3.4 com a abordagem da programação estruturada é portanto eliminada de uma maneira natural.

Na Figura 8.3 apresentamos o diagrama completo com todas as classes do BDA. Devido a limitações de espaço omitimos os atributos e operações das classes. O CD que acompanha a tese traz o modelo completo em UML do projeto, ao qual poderá recorrer o leitor interessado em maiores detalhes.

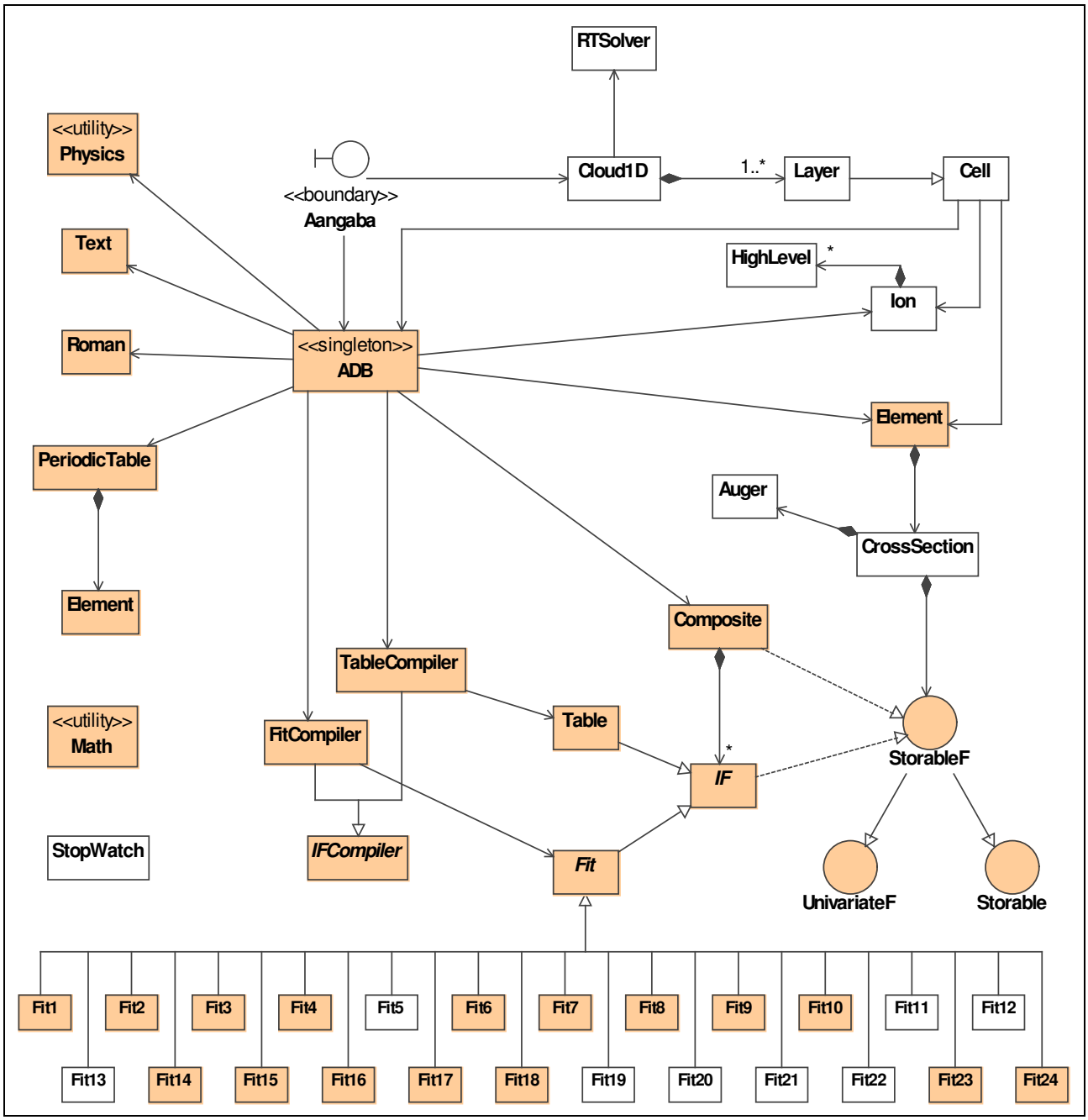


Figura 8.3: Projeto completo do BDA.
Apenas as classes em ocre foram projetadas em detalhes e efetivamente implementadas.

Na verdade o diagrama de classes da Figura 8.3 se refere a um código de fotoionização inteiro, do qual implementamos apenas as classes que constituem o BDA, indicadas em ocre. Apenas as classes integrantes do BDA foram projetadas em detalhes, sendo que as demais encontram-se apenas esquematizadas e deverão ser completadas apenas quando formos incluir as

funcionalidades necessárias para compor um código de fotoionização completo. Evidentemente durante o processo de extensão surgirão também novas classes ainda não identificadas, as quais irão interagir com as classes esquematizadas.

Note que esta abordagem iterativa é recomendada por métodos contemporâneos de desenvolvimento de *software*, nos quais a aplicação é expandida incrementalmente em ciclos de análise/projeto/implementação/testes (Larman 2002). Assim, não apenas não é necessário projetarmos todo o nosso sistema antes de começarmos a implementá-lo, como este procedimento também não é desejável.

Voltando à Figura 8.3, podemos identificar as classes básicas do BDA discutidas anteriormente.

As classes `Fitn` implementadas foram apenas aquelas necessárias para dar suporte aos ajustes das taxas de troca de carga e ionização colisional reunidos no capítulo 4. Outras fórmulas incluídas se referem a seções de choque de fotoionização ou coeficientes de recombinação, dados estes que não foram incorporados ao BDA, como vimos no capítulo 5. Também devido a este motivo não foi necessário implementar as classes `CrossSection` e `Auger` nesta versão do BDA.

Podemos notar que fizemos uma pequena simplificação nos diagramas anteriores, pois as classes `Composite` e `IF` não estendem diretamente a classe `UnivariateF`. Por ser uma classe abstrata pura esta última é projetada como uma interface, a qual é combinada com a interface `Storable` antes de ser implementada pelas classes `Composite` e `IF`. A interface `Storable` especifica os métodos `readObject()` e `writeObject()`, e é implementada por classes cujas instâncias podem ser lidas e escritas no disco. Com isso o nosso projeto possibilita que as taxas do BDA possam ser armazenas em memória permanente.

A fim de preservar as classes `Table` e `Fit` simples e coesas, é preciso mantê-las livres do encargo de compilar os arquivos de dados atômicos. Esta tarefa é melhor incorporada em classes dedicadas, `TableCompiler` e `FitCompiler`, as quais se encarregam de juntar os dados dispostos nos inúmeros *layouts* possíveis (vide Tabela 3.1) e criar as instâncias das classes `Table` e `Fit`. Funcionalidades comuns às classes `TableCompiler` e `FitCompiler` são extraídas na superclasse `IFCompiler`.

Todos os dados atômicos necessários ao cálculo do equilíbrio de ionização de um dado elemento são reunidos na classe `Element`. Como o nosso BDA abrange todos os elementos do H ao Zn, ele terá então 30 instâncias desta classe. A classe `Element` também implementa a interface

Storable, e os seus métodos readObject()/writeObject() simplesmente chamam os métodos correspondentes das inúmeras taxas individuais ("instâncias" de **StorableF**[F]).

A classe **ADB** (de *atomic database*) tem a função de gerenciar o processo de compilação. Inicialmente ela interpreta as palavras-chave nas primeiras linhas dos blocos de dados atômicos e em seguida ativa as classes **TableCompiler** ou **FitCompiler** para compilar os dados em instâncias de **IF**. Em seguida, ela consulta as instâncias de **Element** para verificar se uma dada taxa já se encontra incluída ou não. Caso afirmativo, ela cria uma instância de **Composite** e adiciona as duas contribuições individuais. Obviamente se a taxa incluída já é uma função composta ela simplesmente chama o método add() da classe **Composite** para adicionar uma nova componente.

Outra incumbência da classe **ADB** é a de prover acesso ao BDA para o restante da aplicação. Para que todos os pontos do código utilizem o mesmo conjunto de dados atômicos, é necessário que o acesso se dê através da mesma instância de **ADB**, o que é garantido pelo padrão *Singleton* (Gamma, et al. 1995).

Dependendo da tarefa que esteja executando, a classe **ADB** pode se encontrar em estados distintos, como mostra a Figura 8.4 . Assim que uma instância é criada ela assume um estado de espera Init, no qual ela aguarda a definição da aplicação para que ela inicie a compilação de um novo conjunto de dados atômicos, ou que ela carregue um BDA armazenado em disco e o torne acessível. Se neste estado ela receber a mensagem compileProcess() ela compila o arquivo de dados atômicos indicado (passado como parâmetro) e assume o estado Compiling, no qual pode receber novas mensagens compileProcess() para que sejam adicionados mais dados, ou então a mensagem storeADB() para que o conjunto de dados compilado seja armazenado (ou descartado, se a mensagem cleanUp() for enviada).

Para que um BDA compilado seja carregado devemos enviar a mensagem loadADB() quando a classe estiver no estado Init. Ela então assumirá o estado Loaded, a partir do qual poderá receber mensagens getElements(), e com isso disponibilizar o BDA para outros pontos da aplicação.

Os estados Init, Compiling e Loaded são sub-estados do estado Active, no qual a instância está alocada na memória. Ela poderá ser destruída a partir de qualquer um destes sub-estados invocando-se o destrutor finish().

Neste ponto encerramos a nossa discussão sobre as classes básicas que compõem o nosso projeto do BDA. Na seção seguinte introduzimos as outras classes, que desempenham apenas papéis secundários, e também o esquema esboçado para compor um código de fotoionização.

[F] Evidentemente interfaces, assim como classes abstratas, não podem ser diretamente instanciadas. Em contextos como este estamos nos referindo a *instâncias indiretas* (Rumbaugh et al. 1999), que são instâncias de subclasses manipuladas polimorficamente através da superclasse.

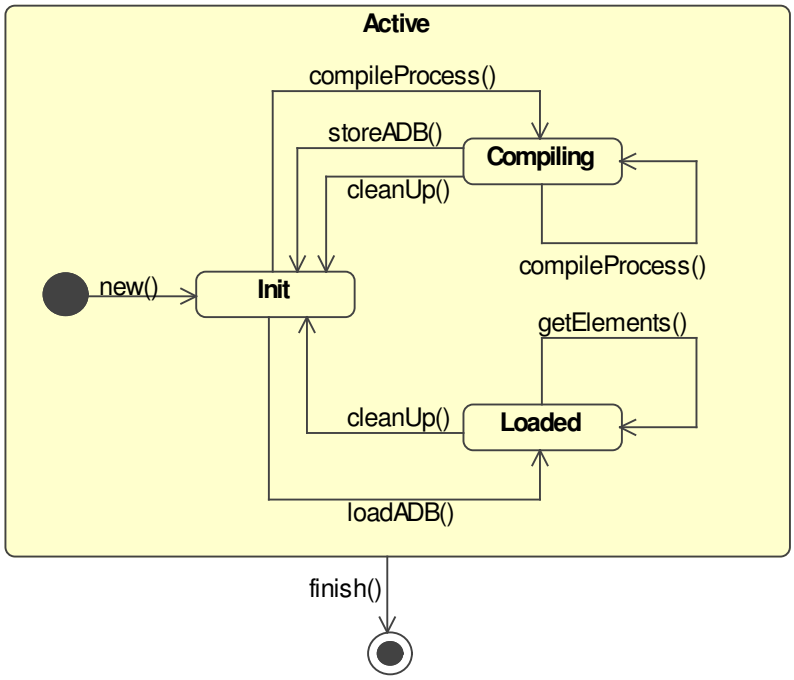


Figura 8.4: Diagrama de estados da classe ADB.

8.2. Outras classes

As classes do BDA estão agrupadas em três pacotes principais: `util`, `fits` e `atomic`.

As classes do pacote `util` são componentes que podem ser reutilizadas em outros contextos mais gerais, não se restringindo ao domínio específico do nosso BDA. Além das interfaces **`Storable`**, **`UnivariateF`**, **`StorableF`**, e das classes **`IF`**, **`Table`**, **`Fit`**, **`Composite`**, vistas na seção anterior, este pacote inclui também as seguintes classes:

- **`Physics`**: esta é uma classe *utility* (Rumbaugh, et al. 1999), ou seja, apenas uma coleção de atributos e métodos estáticos reunidos por conveniência. Neste caso temos apenas constantes fundamentais da física, tais como a velocidade da luz, a constante de Planck, etc., além de constantes úteis para conversão de unidades.

- **`Math`**: também uma classe *utility*. Contém constantes matemáticas e uma função que calcula a integral exponencial $E_1(x)$ (apêndice B).

- **`Text`**: esta classe implementa algumas funcionalidades básicas para manipulação de *strings* ausentes em Fortran, como conversão entre maiúsculas e minúsculas e separação de palavras individuais numa frase (intercaladas por espaços em branco). Ela é útil para processar as palavras-chave dos arquivos de dados atômicos.

- `Roman`: faz a conversão entre números romanos e arábicos. Números romanos aparecem na notação empregada em astronomia para especificar os íons: CIV, SiVIII, etc.

- `PeriodicTable`: representa a tabela periódica dos elementos. No BDA esta classe é útil para converter os símbolos químicos dos elementos para os seus números atômicos.

- `Element`: armazena as propriedades de um dado elemento da tabela periódica, como o símbolo químico, o número atômico e o nome completo. Não confundir com a classe homônima vista anteriormente, a qual se encontra no pacote `atomic`.

- `StopWatch`: cronômetro para medir tempos de execução de trechos de código, para fins de otimização.

O pacote `fits` reúne as classes `Fitn`, com os ajustes que implementam as fórmulas matemáticas empregadas no BDA.

O pacote `atomic` engloba as classes diretamente relacionadas ao BDA, tais como as classes `IFCompiler`, `TableCompiler`, `FitCompiler`, `Element` e `ADB` vistas anteriormente. Além destas, temos também:

- `CrossSection`: representa a seção de choque de fotoionização de um dado íon, que é uma função da energia.

- `Auger`: armazena as probabilidades de ejeção de elétrons Auger associados à fotoionização de uma determinada camada.

- `Ion`: modela um determinado átomo/íon com seus vários níveis de energia. Engloba todas as taxas ligado-ligado necessárias para o cálculo do equilíbrio estatístico destes níveis. Esta classe é o análogo em programação orientada a objetos do código PopRatio, discutido na seção 2.3 . Porém, o código PopRatio pode lidar apenas com um único átomo/íon de cada vez, ao passo que aqui podemos criar várias instâncias separadas desta classe e lidar com o equilíbrio de vários átomos/íons simultaneamente. Além disso, a incorporação desta funcionalidade no arcabouço desenvolvido do BDA, juntamente com a adoção de um formato semelhante ao apresentado no capítulo 3 para os arquivos de dados atômicos de entrada, traria muito mais flexibilidade à aplicação.

- `HighLevel`: representa os níveis de fluorescência associados a um determinado íon. É responsável pelo cálculo das taxas de fluorescência através deste níveis.

O restante das classes na Figura 8.3 estão incluídas apenas para colocar o BDA no contexto maior de um código de fotoionização.

A classe `Cell` representa uma dada região do espaço de condições físicas constantes. Ela seria o coração de qualquer código de fotoionização, uma vez que todos eles adotam basicamente a mesma estratégia de aproximar a nuvem como uma grade de regiões finitas. Sua função é a de armazenar e calcular as quantidades físicas relevantes nessas regiões, tais como temperatura, opacidade, emissividade, etc.

A ação central do código de fotoionização se desenrola a partir do momento em que as instâncias da classe `Cell` recebem a mensagem `equilibrate()`, com o campo de radiação passado como um parâmetro. A classe `Cell` então coordena e divide o trabalho entre outras abstrações que modelam os diversos constituintes do meio, cada uma se encarregando do aspecto do cálculo no qual são especializadas. Por exemplo, a classe `Element` acima se encarregaria de efetuar o equilíbrio de ionização na célula, ao passo que o balanceamento estatístico dos níveis de energia ficaria a cargo da classe `Ion`. Estas duas classes também calculariam a contribuição individual dos inúmeros elementos (C, N, O, etc.) e íons (C^+,O^0, Si^+, etc.) para o equilíbrio térmico, a opacidade e a emissividade da célula.

Este modelo pode ser facilmente estendido para englobar outros constituintes ao meio da nuvem. Suponhamos por exemplo que desejamos considerar moléculas e grãos em nossa simulação numérica. Tudo o que temos a fazer é criar uma nova classe `Molecule`, que se encarregaria de efetuar o equilíbrio químico das moléculas, e uma classe `Grain`, que se ocuparia de calcular as propriedades dos grãos.

A classe `Layer` estende a classe `Cell` acrescentando atributos como distância à fonte central de radiação e espessura, para que esta deixe de representar apenas uma região genérica do espaço e passe a encarnar camadas esféricas concêntricas no âmbito de um código de fotoionização unidimensional.

Cada instância da classe `Layer` é um sistema isolado que conhece apenas a região que representa, e não tem acesso a nenhuma informação acerca do que ocorre nas outras partes da nuvem. No entanto, as condições físicas em cada região estão acopladas às condições físicas do restante da nuvem através do campo de radiação. Cabe então à classe `Cloud1D` coordenar o transporte de radiação entre as camadas, em colaboração com a classe auxiliar `RTSolver` (de *radiative transfer solver*), que é altamente especializada em resolver a equação diferencial específica do transporte radiativo.

A classe `Cloud1D` também se encarrega de resolver adequadamente a grade do modelo, acrescentando mais camadas em regiões onde as condições físicas variam mais rapidamente. Ela também armazena as propriedades características da nuvem como um todo, tais como abundâncias, densidade total, campo de radiação incidente, etc.

As componentes `Cell` e `RTSolver` são altamente reutilizáveis. Para criar um código de fotoionização tridimensional, por exemplo, podemos criar uma classe `Cube` a partir de `Cell` e agregá-la numa classe `Cloud3D`, a qual também se comunicaria com `RTSolver`.

Por fim, a classe `Aangaba` representa o código de fotoionização como um todo. Ela é dita uma classe *boundary* por ser a fronteira do nosso sistema, e seria uma porta de comunicação com outras camadas da aplicação, como uma interface gráfica por exemplo. Por definir um ponto de acesso único ao nosso sistema, esta é uma instância do padrão *Façade* (Gamma, et al. 1995).

Agora que introduzimos todas as classes individuais do nosso projeto, podemos ter uma visão mais ampla de todo o conjunto. O código de fotoionização proposto está estruturado em módulos coesos, numa arquitetura em camadas como mostra a Figura 8.5 .

A "camada básica" é composta pelas classes genéricas do pacote `util`, sendo que o bloco "ajustes" é meramente uma extensão dessa camada que contém as classes do pacote `fits` ou ainda outras fórmulas matemáticas definidas pelo usuário.

Acima da camada básica temos uma camada para gerenciar o banco de dados atômicos. Além da classe `ADB`, esta camada engloba as classes encarregadas de compilar os arquivos de dados atômicos (`IFCompiler` e subclasses) e as classes relacionadas aos processos atômicos em si, tais como `CrossSection`, `Auger`, `HighLevel`, etc.

A camada "núcleo" contém as classes que realizam as tarefas fundamentais em um código de fotoionização. Aqui incluímos a classe `Cell` e outras classes que representam os constituintes do meio, como as classes `Element`, `Ion`, `Molecule`, `Grain`, etc.

Figura 8.5: Arquitetura em camadas de um código de fotoionização.

Finalmente, no topo da arquitetura temos uma camada cuja função é a de calcular o transporte de radiação. Aqui colocamos as classes `Layer`, `Cloud1D` e `RTSolver` e outras que irão surgir ao refinarmos o nosso modelo.

Posteriormente o projeto do nosso código de fotoionização poderá ser ainda estendido para incluir outros ingredientes, como a hidrodinâmica e a gravidade, organizados em camadas adicionais em nossa arquitetura.

Os blocos arquiteturais acima podem ser utilizados por outros autores para compor as suas próprias aplicações. No futuro um pesquisador interessado em realizar uma simulação numérica de uma estrela binária circundada por um disco de acresção em forma de toro, por exemplo, não necessitará construir totalmente o seu código desde a estaca zero. Ele poderá partir da infra-estrutura básica do nosso código de fotoionização, integrá-la com componentes desenvolvidos por outros autores e criar apenas as partes que lidam com os aspectos mais específicos do seu domínio.

Ao deixar de reinventar a roda, este autor hipotético não apenas chegaria ao seu código final mais rapidamente, como este também teria uma qualidade superior e uma garantia maior de resultados corretos sem a influência de *bugs*. Acreditamos que a programação orientada a objetos é uma ferramenta essencial para possibilitar a criação de componentes em *software* altamente modularizadas e reutilizáveis quando estamos lidando com códigos científicos extremamente complexos.

8.3. A implementação

O modelo em UML especifica o projeto do BDA de maneira independente de qualquer linguagem de programação específica, de modo que estamos livres para implementá-lo em qualquer linguagem que suporte programação orientada a objetos.

Conforme já adiantamos em capítulos anteriores, optamos por implementar o BDA em Fortran, a fim de facilitar a sua integração com o código de fotoionização Aangaba. As classes em ocre na Figura 8.3 foram então codificadas empregando o processo de emulação dos conceitos de programação orientada a objetos discutido no capítulo anterior, o que no final resultou em pouco mais de 10.000 linhas de código.

Infelizmente nenhuma ferramenta CASE dá suporte à geração de código em Fortran, pelo que todo o projeto teve que ser codificado manualmente desde o início. Assim, embora dispuséssemos da ferramenta Magic Draw (MagicDraw) para a elaboração dos diagramas em UML, ela não pôde ser utilizada nesta fase para geração automática de código ou ainda para manter o modelo em UML sincronizado com o código à medida que ele evoluia.

No momento não nos preocupamos em implementar nenhuma das funcionalidades avançadas discutidas no capítulo 3, tais como otimização dos dados ou testes de sanidade e completeza. Esta versão inicial do BDA apenas compila os arquivos HTML reunindo as taxas num

único arquivo binário, tal qual foram entradas pelo usuário. Também nem todos os *layouts* especificados na Tabela 3.1 estão suportados: apenas aqueles utilizados nos arquivos das taxas de troca de carga e ionização colisional foram implementados.

A fim de assegurar a qualidade do código implementado, é necessário testá-lo o máximo possível. Assim, para cada classe recém-codificada nos preocupamos também em elaborar uma bateria de testes para verificar a correção de suas funcionalidades.

Para que os testes sejam executados freqüentemente é preciso que eles sejam auto-verificáveis, ou seja, eles próprios se encarreguem de averiguar se tudo transcorreu conforme o esperado. O ideal portanto é que ao final da execução da bateria de testes o programador seja informado apenas se um determinado teste falhou ou não.

No esquema de testes que elaboramos, um teste individual é implementado simplesmente como uma função booleana que retorna `true` se o teste passou, e `false` caso contrário. Uma bateria de testes é composta por uma função booleana que chama todos os testes individuais e retorna `true` apenas se todos eles passaram. Cada classe tem a sua própria bateria de testes, as quais são reunidas numa bateria de testes maior contendo as baterias de testes de todas as classes de um determinado pacote. No topo desta hierarquia está a bateria de testes que engloba as baterias de testes de todos os pacotes da aplicação.

A Figura 8.6 mostra os resultados exibidos na tela após a execução dos testes. As mensagens de diagnóstico estão organizadas em parágrafos, cada um deles se referindo à bateria de testes de um determinado pacote. A última linha de cada parágrafo informa se todas as baterias de testes daquele pacote foram bem-sucedidas, e as linhas acima se referem às baterias de cada classe integrante do pacote. Porém, tudo o que o usuário necessita fazer ao rodar os testes é inspecionar apenas a última linha da tela para averiguar se todos os testes obtiveram êxito. Apenas caso algum teste tenha falhado será preciso rolar a tela até as últimas linhas de cada parágrafo para descobrir em qual pacote se encontra o problema, e então mais acima para identificar exatamente quais foram os testes que deram errado.

Pudemos comprovar na prática que os testes automatizados são uma maneira efetiva de apanhar *bugs* no código tão logo eles sejam introduzidos, o que acelerou em muito a etapa de implementação, pois quanto mais tardiamente um *bug* é detectado, mais caro será corrigi-lo.

Entretanto, a sua importância maior virá quando posteriormente modificarmos o nosso código. Já mencionamos que processos modernos de desenvolvimento de *software* recomendam que as funcionalidades da aplicação sejam implementadas gradualmente de maneira iterativa, sem que seja necessário projetá-la em sua totalidade desde o início. Os testes automatizados são fundamentais para que futuramente possamos alterar o nosso projeto sem medo de introduzir *bugs* e quebrar o código que está funcionando corretamente. Uma técnica efetiva para melhorar o projeto

de códigos existentes, conhecida como *refatoração* (Fowler 2000), se apóia na existência dessas baterias de testes automatizados que são sempre executados ao final de pequenas mudanças incrementais.

```
"D:\dados\NewAangaba\ADT\Debug\ADT.exe"
 Element: passed all tests.
 PeriodicTable: passed all tests.
 Text: passed all tests.
 Roman: passed all tests.
 Table: passed all tests.
 PolyFit: passed all tests.
 PolyIF: passed all tests.
 Composite: passed all tests.
 StorableF: passed all tests.
 util: passed all tests.

 Fit1: passed all tests.
 fits: passed all tests.

 Element: passed all tests.
 FitCompiler: passed all tests.
 TableCompiler: passed all tests.
 ADB: passed all tests.
 atomic: passed all tests.

 CONGRATULATIONS: everything is working fine! :-)
Press any key to continue_
```

```
"D:\dados\NewAangaba\ADT\Debug\ADT.exe"
 Element:test1: FAILED
 Element: FAILED.
 PeriodicTable:test1: FAILED
 PeriodicTable:test2: FAILED
 PeriodicTable:test3: FAILED
 PeriodicTable: FAILED.
 Text: passed all tests.
 Roman: passed all tests.
 Table: passed all tests.
 PolyFit: passed all tests.
 PolyIF: passed all tests.
 Composite: passed all tests.
 StorableF: passed all tests.
 util: FAILED.

 Fit1: passed all tests.
 fits: passed all tests.

 Element: passed all tests.
 FitCompiler: passed all tests.
 TableCompiler: passed all tests.
 ADB: passed all tests.
 atomic: passed all tests.

 SORRY, there are bugs to be fixed... :-(
Press any key to continue_
```

Figura 8.6: Telas exibidas após a execução da bateria de testes.
Na tela de cima todas as baterias de testes alcançaram êxito; na tela de baixo introduzimos intencionalmente um *bug* na classe `Element` do pacote `util`, o que acabou também quebrando os testes da classe `PeriodicTable`.

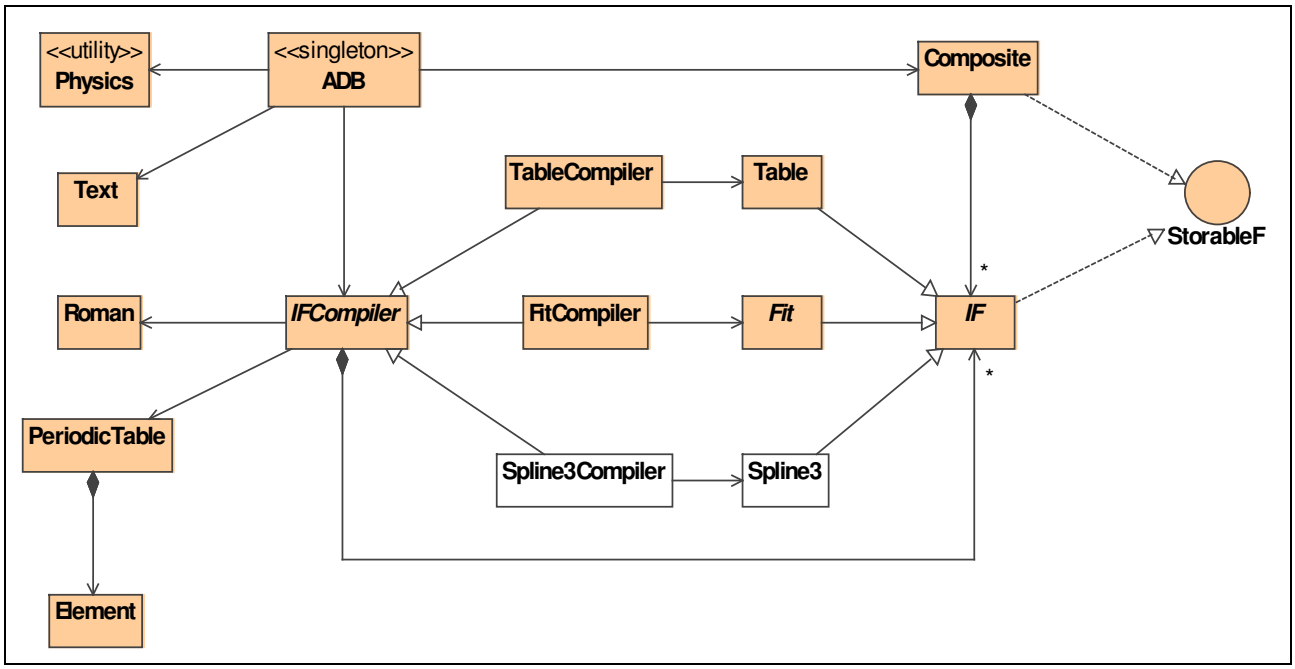


Figura 8.7: Refatoração sugerida para melhorar o projeto do BDA.

Uma sugestão para melhorar o projeto do BDA apresentado na Figura 8.3 que ainda gostaríamos de deixar aqui, seria refatorá-lo para o projeto da Figura 8.7 .

No projeto original a classe `ADB` está muito acoplada com outras classes do BDA, sendo que o acoplamento com todas as subclasses de `IFCompiler` em particular é um obstáculo à introdução de novos tipos de funções de uma variável. No novo projeto, a classe ADB manipula as instâncias de `TableCompiler` e `FitCompiler` unicamente através de suas interfaces em comum em `IFCompiler`, o que nos permite introduzir novos tipos de funções com um mínimo de impacto. No exemplo da Figura 8.7 adicionamos uma nova classe que representa uma função *spline* cúbica.

Note também que no novo projeto o acoplamento da classe `ADB` foi reduzido ainda um pouco mais pela transferência das associações com as classes `Roman` e `PeriodicTable` para a classe `IFCompiler`, a qual pode assumir a responsabilidade de processar os nomes dos íons através de um método `parseName()` que seria herdado em todas as suas subclasses.

Referências

Fowler, M. 2000, *"Refactoring"* (Addison-Wesley)

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. 1995, *"Design Patterns"* (Addison-Wesley)

Larman, C. 2002, *"Applying UML and patterns"*, 2ª ed. (Upper Saddle River, NJ: Prentice Hall PTR)

MagicDraw, Ferramenta CASE para elaboração de diagramas em UML e geração de código: http://www.magicdraw.com

Rumbaugh, J., Jacobson, I., & Booch, G. 1999, *"The Unified Modeling Language Reference Manual"* (Addison-Wesley)

9. Os Sistemas Lyman-Limit

9.1. Introdução

Neste capítulo iremos aplicar o código de fotoionização Aangaba aos sistemas em absorção dos QSOs a fim de calcular as razões entre as densidades de coluna dos níveis de estrutura fina. Com isto estaremos atingindo o objetivo final que motivou este trabalho que é o tratamento do equilíbrio estatístico dos íons C^0, C^+, O^0, Si^+ e Fe^+ considerando detalhadamente a variação das condições físicas camada a camada ao longo da nuvem.

Existem na literatura inúmeros sistemas DLA com as linhas de estrutura fina em absorção detectadas com sucesso (veja, por exemplo, em Silva 1999; Silva & Viegas 2002). Infelizmente não podemos analisá-los neste momento devido ao fato de o código Aangaba não incluir moléculas, as quais podem ser uma fonte importante de resfriamento nestes sistemas (em particular, a molécula de H_2). Por esse motivo elegemos os sistemas LL como objeto de nosso estudo.

Além disto, densidades de coluna de hidrogênio neutro tão altas quanto as encontradas nos sistemas DLA implicam em temperaturas predominantemente baixas, o que acarreta dificuldades de convergência nos modelos. Nos deparamos com problemas deste tipo mesmo com os sistemas LL de densidades de coluna mais altas e modelos muito densos, conforme veremos adiante.

As linhas de estrutura fina em absorção nos sistemas LL surgem em conexão com um problema interessante relacionado à fonte de ionização destes sistemas. A hipótese geralmente aceita atribui a fonte de radiação ao campo extragalático produzido pelos QSOs de fundo (Haardt & Madau 1996), o que no entanto conduz a certas dificuldades.

Observações de densidades de coluna de elementos em vários estágios de ionização permitem impor restrições ao parâmetro de ionização dos modelos:

$$U = \frac{\Phi_{\mathrm{H}}}{c\, n_{\mathrm{H}}} \quad , \tag{9.1}$$

onde Φ_{H} é o fluxo de fótons capazes de ionizar o hidrogênio, c é a velocidade da luz e n_{H} é a densidade volumétrica total de hidrogênio no meio.

Modelos com o campo integrado dos QSOs como fonte de radiação requerem $U \simeq 10^{-2}$ para explicar as densidades de coluna observadas, o qual substituído em (9.1) juntamente com o valor de

$\Phi_{\rm H}$ resulta em $n_{\rm H} < 10^{-2}$ cm^{-3}. Densidades baixíssimas desta ordem implicam numa região absorvente enorme, com tamanhos característicos $l \simeq 20$ kpc (Gruenwald & Viegas 1993). Porém, se o tamanho destas regiões é semelhante ao tamanho dos aglomerados globulares da Galáxia, então a densidade de hidrogênio deveria ser $n_{\rm H} \simeq 10$ cm^{-3}, e os modelos de fotoionização com o campo integrado dos QSOs não explicariam as densidades de coluna observadas a menos que uma fonte adicional local de radiação esteja presente.

Um modelo alternativo que procura explicar as observações identifica os sistemas LL como condensações frias imersas num halo quente na galáxia, o qual desempenha o papel de fonte local de ionização (Viegas & Friaça 1995). Um halo quente é previsto nos estágios iniciais da evolução da galáxia pelo modelo químeo-dinâmico (Friaça & Terlevich 1998).

As linhas de estrutura fina podem ajudar a esclarecer a questão da fonte de ionização dos sistemas LL, uma vez que elas fornecem uma determinação independente da densidade.

Um sistema que impõe fortes vínculos aos modelos de fotoionização são os sistemas LLs na direção do QSO HS 1700+6416. Neste objeto foram combinados espectros ultra-violeta obtidos com o telescópio espacial Hubble com observações no óptico a partir do chão para derivar densidades de coluna dos elementos CNO em inúmeros estágios de ionização: O^{+2} - O^{+5}, N^{+} - N^{+4} e C^{+} - C^{+3} (Vogel & Reimers 1993).

Não é possível explicar as razões de densidades de coluna observadas nesses sistemas assumindo o campo integrado dos QSOs como única fonte de ionização. Posteriormente, a detecção de He^{0} impossibilitou os autores de construírem qualquer modelo que desse conta de explicar simultaneamente a densidade de coluna do He^{0} e as densidades de coluna dos metais (Reimers & Vogel 1993). O modelo do halo quente, no entanto, foi bem-sucedido ao ser aplicado a estes sistemas (Viegas & Friaça 1995).

Pelo que sabemos, atualmente o único sistema LL no qual foi possível detectar as linhas de estrutura fina é o sistema em $z_{\rm a} = 2.652$ na direção de Q2231-00 (Prochaska 1999). Infelizmente viemos a descobrir que este sistema não se presta a nenhuma análise quantitativa, pelas razões que expomos na seção seguinte.

A abordagem que adotamos consistiu então em produzir uma grade de modelos de fotoionização sob condições físicas representativas de sistemas LL, a qual poderá ser aplicada a novos sistemas que venham a ser observados. Acreditamos que o quadro atual de falta de dados é passageiro, e que novas observações em espectrógrafos de alta resolução em telescópios de grande porte deverão surgir na literatura num futuro próximo. Nossa opinião é justificada pelo fato de os sistemas LL consistirem num terreno amplo e inexplorado, sendo o caminho natural a seguir uma vez que os sistemas DLA foram muito estudados.

9.2. O sistema em z_a = 2.652 na direção de Q2231-00

Este sistema foi estudado detalhadamente por Prochaska (1999) com base num espectro de alta resolução obtido no telescópio Keck I.

O autor dividiu o perfil das linhas em absorção em quatro componentes, nas quais foi possível determinar as densidades de coluna para os íons listados na Tabela 9.1 . Utilizando o código de fotoionização Cloudy, ele derivou o parâmetro de ionização a partir das razões *N*(Fe II) / *N*(Fe III) e *N*(Si II) / *N*(Si IV). A determinação da densidade volumétrica com base na linha de estrutura fina do C II, aliada ao modelo de fotoionização, foram o ponto de partida para a determinação das condições físicas da nuvem e outras quantidades (metalicidade, tamanho característico, intensidade do campo de radiação, etc.).

O tratamento dedicado às linhas de estrutura fina foi extremamente simplificado: a razão de povoamento dos níveis foi obtida por intermédio de uma fórmula analítica grosseira, e empregando densidades médias ao longo da nuvem. Por esse motivo seria interessante repetirmos a modelagem teórica com o procedimento aqui proposto de resolver numericamente as equações exatas de balanceamento estatístico com as condições físicas variando camada a camada da nuvem.

No entanto, existem problemas fundamentais com esta observação.

Primeiramente podemos notar na Tabela 9.1 que as observações fornecem apenas limites inferiores para as densidades de coluna totais do Si II e Si IV, os quais o autor estranhamente utiliza para derivar um limite superior na razão *N*(Si II) / *N*(Si IV)!

Íon	λ_0 [Å]	região 1	região 2	região 3	região 4	Total
H I	1215	...	...	...	...	19.12 ± 0.2
C II	1334	14.24 ± 0.01	15.41 ± 0.08[☞]	> 14.76	14.42 ± 0.03	> 15.41
C II*	1335	...	12.90 ± 0.03	...	...	...
O I	1302	14.49 ± 0.02	> 15.20	14.22 ± 0.02	13.82 ± 0.02	> 15.33
Al II	1670	12.35 ± 0.01	> 13.80	12.88 ± 0.02	12.67 ± 0.02	> 13.84
Si II	1304	13.58 ± 0.02	> 14.87	14.18 ± 0.01	14.03 ± 0.02	> 15.02
Si IV	1402	...	> 14.50	> 14.21	> 14.09	> 14.78
Fe II	1608	12.96 ± 0.08	14.34 ± 0.01	13.38 ± 0.03	13.43 ± 0.03	14.44 ± 0.05
Fe III	1122	13.71 ± 0.04	14.69 ± 0.03	13.86 ± 0.03	13.83 ± 0.03	14.84 ± 0.02
Ni II	1741	...	...	...	...	13.36 ± 0.04

Tabela 9.1: Densidades de coluna no sistema em z_a = 2.652 na direção de Q2231-00.

Os valores correspondem aos logaritmos das densidades de coluna (em cm^{-2}). Extraído de Prochaska (1999).

[☞] Derivado a partir de *N*(FeII) com base na razão *N*(C II)/*N*(FeII) observada na região 4.

Um outro problema muito mais grave vem do fato de as linhas essenciais à análise caírem na região da floresta de Ly-α do espectro. A condição para que uma linha metálica de comprimento de onda λ_Z não caia na floresta pode ser expressa como:

$$\lambda_Z(1+z_a) > \lambda_{Ly\text{-}\alpha}(1+z_e) \quad , \tag{9.2}$$

a qual, após a substituição do comprimento de onda da transição de Ly-α $\lambda_{Ly\text{-}\alpha} = 1215$ Å, e dos *redshifts* de absorção e emissão tirados do artigo $z_a = 2.652$ e $z_e = 3.02$, resulta em:

$$\lambda_Z > 1337 \text{ Å} \quad , \tag{9.3}$$

condição esta que é violada pela maioria das linhas na Tabela 9.1, incluindo-se as do Si II e Fe III, nas quais o autor se baseou para inferir o parâmetro de ionização.

É curioso notar que em todo o artigo a questão da contaminação pelas linhas da floresta sequer é mencionada, sendo que até mesmo a linha de estrutura fina proveniente do estado excitado é afetada! Por ser uma transição fraca, esta linha estaria sujeita a forte contaminação pela floresta, pelo que a determinação da densidade de coluna do C II* deveria, quando muito, ser tomada como um limite superior apenas.

Ao nosso ver, estes são obstáculos sérios que invalidam quaisquer conclusões obtidas no artigo e também desqualificam esta observação para qualquer análise mais aprofundada.

9.3. A grade de modelos

O primeiro passo para a aplicação do código de fotoionização aos sistemas LL consistiu em modificar o código Aangaba a fim de que ele utilize a infra-estrutura do banco de dados atômicos desenvolvido, descrito no capítulo anterior. Em particular, as rotinas que calculam as taxas de troca de carga e ionização colisional foram alteradas para que consultem o BDA, o qual por sua vez se encarrega de devolver as taxas reunidas no capítulo 4.

Outras modificações também se fizeram necessárias para dar suporte ao campo integrado dos QSOs. Em nossa grade de modelos consideramos dois campos possíveis: um modelo de halo quente com idade t = 0.206 Gano e distância ao centro da galáxia r = 30 kpc (Viegas & Friaça 1995), e o campo integrado dos QSOs calculado com um parâmetro de desaceleração q_0 = 0.1 (Haardt & Madau 1996). Estes dois campos estão plotados na Figura 9.1; o leitor é remetido a Silva (1999) para gráficos semelhantes destes campos para outros conjuntos de parâmetros.

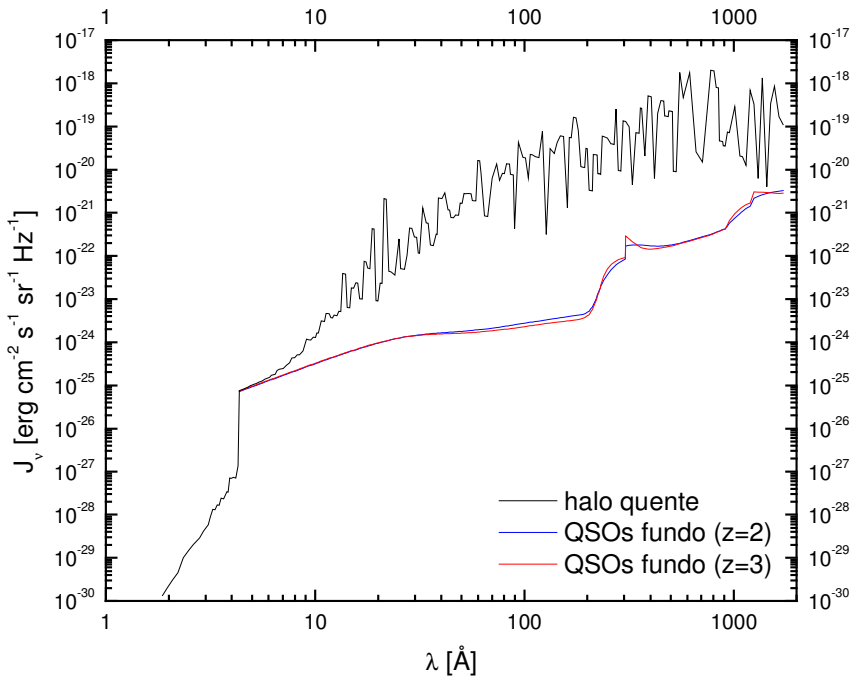


Figura 9.1: Campo de radiação do halo quente e integrado dos QSOs.

O campo do halo quente refere-se a um modelo com idade t = 0.206 Gano e distância ao centro da galáxia r = 30 kpc; o campo integrado dos QSOs corresponde a um parâmetro de desaceleração de q_0 = 0.1 .

Nossos modelos assumem uma nuvem homogênea de densidade constante, aproximando-a como placas plano-paralelas de condições físicas constantes. No Aangaba, a geometria plano-paralela é emulada tomando-se o raio inicial da nuvem muito maior do que a sua espessura.

Para melhor reproduzir uma região iluminada externamente com o campo de radiação incidindo sobre seus dois lados, o modelo computado pelo Aangaba é tomado como sendo apenas metade da nuvem, sendo a outra metade a imagem refletida desta.

Uma vez que os sistemas LL estão consideravelmente ionizados, as densidades de coluna dos íons de baixa ionização que estamos interessados em calcular estarão concentradas nas camadas mais externas da nuvem. Para assegurar que estas regiões estejam suficientemente resolvidas, o tamanho dos passos é controlado de forma que nenhuma das densidades de coluna calculadas esteja acumulada em mais do que 20% numa única camada.

As condições físicas ao longo da nuvem são obtidas percorrendo-se o modelo de fotoionização de fora para dentro na metade calculada, com o cuidado de que ao atingirmos a densidade de coluna de hidrogênio neutro mais baixa na qual estamos interessados tenhamos pelo menos 100 camadas. Este critério resulta num número excessivo de camadas ao atingirmos densidades de coluna mais altas, pelo que é necessário calcular dois ou três modelos de fotoionização (variando-se os parâmetros que controlam o tamanho do passo) e juntá-los para que possamos percorrer todo o intervalo de densidades de coluna de hidrogênio neutro desejado.

	número	dex
H	1.00E+00	12.00
He	9.77E-02	10.99
C	3.63E-04	8.56
N	1.12E-04	8.05
O	8.51E-04	8.93
Ne	1.23E-04	8.09
Mg	3.80E-05	7.58
Si	3.55E-05	7.55
S	1.86E-05	7.27
Cl	1.86E-07	5.27
Ar	3.63E-06	6.56
Fe	3.24E-05	7.51

Tabela 9.2: Abundâncias do sistema-solar.
A primeira coluna dá a abundância por número do elemento, sendo expressa também nas unidades usuais log(Z/H) + 12.00. Valores extraídos de Anders & Grevesse (1989).

As abundâncias do sistema-solar adotadas foram extraídas de Anders & Grevesse (1989), e estão mostradas na Tabela 9.2 para os elementos considerados pelo Aangaba. A maioria dos valores correspondem a abundâncias meteoríticas, com exceção das abundâncias do C, N e O, que são da fotosfera solar, e do Ne e Ar, que foram baseadas em outros dados astronômicos.

Ao rodar o código de fotoionização, todas as quantidades físicas de interesse são armazenadas num arquivo separado para todas as camadas do modelo. Isso inclui a densidade de coluna de hidrogênio neutro ao longo da nuvem, as espessuras das camadas, as temperaturas cinéticas, as densidades de coluna dos íons para os quais desejamos calcular o povoamento dos níveis de estrutura fina, as densidades volumétricas das principais partículas que podem excitar colisionalmente os níveis de estrutura fina, além das intensidades do campo de radiação. Todos estes dados são digeridos por um segundo programa que se encarrega de chamar o PopRatio para que sejam calculadas as razões de povoamento dos níveis de estrutura fina e também todas as taxas ligado-ligado relevantes, a fim de que possamos inferir quais são os mecanismos de excitação mais importantes em cada caso.

Dos cinco íons apontados na seção 2.1, apenas o C^+, Si^+ e Fe^+ foram considerados. Não incluímos o C^0 porque este teria uma abundância fracional baixíssima em sistemas tão ionizados quanto os LLs, e portanto não seria passível de detecção. O O^0 também foi deixado de fora, pois como apontamos na seção 2.5 incluí-lo em sistemas ionizados exigiria o acoplamento do equilíbrio estatístico com o equilíbrio de ionização, o que está fora do escopo deste trabalho.

Todos os mecanismos de excitação dos níveis de estrutura fina são levados em conta: colisões com partículas do meio, a radiação cósmica de fundo ou ainda o campo de radiação presente na nuvem (primário ou difuso).

O modelo atômico adotado para o C^+ considera colisões com elétrons, átomos de hidrogênio e prótons (este último apenas para $T \geq 1000$ K, vide Figura 2.2). Para $T < 1000$ K são incluídos apenas os dois níveis do termo fundamental no balanceamento estatístico, ao passo que para $T \geq 1000$ K é utilizado o modelo completo de 10 níveis.

Para o Si^+ são consideradas colisões com elétrons (adotando a extrapolação da força de colisão mostrada na Figura 2.3 até $T = 100$ K), átomos de hidrogênio (para $T < 2000$ K) e prótons (para $T > 10.000$ K). Para $\log T < 3.6$ são incluídos apenas os dois níveis do termo fundamental no balanceamento estatístico, ao passo que para $\log T \geq 3.6$ é utilizado o modelo completo de 11 níveis.

Com relação ao Fe^+, o único parceiro de colisão considerado são elétrons. O modelo atômico completo de 16 níveis empregado é válido para $T \geq 1000$ K (vide Figura 2.7).

Para nenhum dos três íons é possível ir além de $T > 40.000$ K: para o C^+ e Si^+ não estão disponíveis taxas de colisões com elétrons a essas temperaturas[☞] e para o Fe^+ pode ser necessário incluir mais de 16 níveis no modelo atômico (vide seção 2.6). Esse limite não influencia os resultados, pois estes íons não estão presentes num gás fotoionizado a essas temperaturas.

A faixa de condições físicas investigadas em nossa grade de modelos abrange densidades de coluna de hidrogênio neutro no intervalo $17 \leq \log N(\mathrm{HI}) \leq 20$ (sendo que na verdade $\log N(\mathrm{HI}) \geq 19$ seria mais característico de sistemas DLA), densidades volumétricas de hidrogênio totais $n_{\mathrm{H}} = 10^{-2}, 10^{-1}, 10^{0}, 10^{1}$ cm^{-3}, *redshifts* z = 2, 3 e metalicidades [Z/H] = 0, -1, -2[☞☞].

Nas seções seguintes analisamos em detalhes os resultados obtidos, separadamente para cada um dos campos de radiação adotados.

Em todos os modelos calculamos as razões de densidades de coluna totais dos níveis de estrutura fina N^*/N, além de todas as taxas ligado-ligado relevantes para que possamos compreender quais são os mecanismos de excitação atuantes em cada camada da nuvem. Também apresentamos aqui as densidades de coluna individuais para o nível de estrutura fina excitado, a fim de que o observador possa avaliar o quão difícil será detectar a linha em absorção proveniente deste nível em seu espectro.

☞ Note que a extrapolação até $T = 10^5$ K da taxa de excitação eletrônica para o C^+ na figura 2.2 foi feita apenas para a transição envolvendo os dois primeiros níveis.

☞☞ $[\mathrm{Z/H}] \equiv \log(\mathrm{Z/H}) - \log(\mathrm{Z/H})_{\odot}$.

Outra informação que desejamos obter é o quão importante é o tratamento camada a camada do equilíbrio estatístico dos níveis de estrutura fina proposto em nosso trabalho. Para tal, comparamos os valores de N^*/N obtidos com a previsão de um "modelo simplificado", no qual calculamos n^*/n utilizando as médias das densidades dos parceiros de colisão e temperatura cinética ao longo da nuvem (ponderadas pela largura de cada camada) e o campo de radiação incidente.

Também calculamos os tamanhos e as massas das regiões absorventes.

9.3.1. O campo integrado dos QSOs

As figuras 9.2-9.4 ilustram os resultados obtidos para as densidades de coluna dos níveis de estrutura fina do C^+, Si^+ e Fe^+ para os modelos ionizados pelo campo integrado dos QSOs.

Todas as curvas das razões de povoamento como funções da densidade de coluna de hidrogênio neutro apresentam o mesmo comportamento qualitativo: um patamar constante no início que começa a decair apenas no final. Quanto maior a densidade, menor será a densidade de coluna a partir de onde se dá essa queda.

Em todos os casos o mecanismo de excitação mais importante são colisões com elétrons. Em baixas densidades de coluna, o meio está totalmente ionizado e a densidade eletrônica é constante, sendo que as variações de temperatura nessas regiões também não são grandes o bastante para produzir diferenças significativas nos povoamentos dos níveis de estrutura fina. A queda nas curvas ocorre somente quando entramos na região de recombinação e a densidade eletrônica começa a cair.

No C^+ colisões com átomos de hidrogênio também podem ser um mecanismo de excitação significativo na região recombinada. Porém, este mecanismo não passa a atuar imediatamente ao atravessarmos a fronteira da região de recombinação, pois é necessário que o gás esteja bastante recombinado para que as taxas de excitação sejam comparáveis às de colisões com elétrons, com $n_{\mathrm{H}^0} > 100\, n_{\mathrm{e}^-}$ (vide seção 2.4). Assim, colisões com átomos de hidrogênio só são importantes bem no final da queda nas curvas das razões de densidades de coluna.

Para o Si^+ as colisões com elétrons são o único mecanismo de excitação importante, e a queda nas razões de densidades de coluna refletem unicamente a diminuição da densidade eletrônica. O mesmo podemos dizer para o Fe^+, com exceção da curva correspondente a $n_{\mathrm{H}} = 10^{-2}$ cm^{-3}, onde a fluorescência é importante no final.

As alterações produzidas nos modelos ao passarmos de $z = 2$ para $z = 3$ são pequenas, uma vez que o campo de radiação integrado dos QSOs a esses *redshifts* é aproximadamente igual (Figura 9.1) e a radiação cósmica de fundo não é um mecanismo de excitação relevante para nenhum dos íons investigados. As únicas exceções são as curvas de baixas densidades para o C^+: com $n_{\mathrm{H}} = 10^{-2}$ cm^{-3} a radiação cósmica de fundo é o principal mecanismo de excitação em $z = 3$, contribuindo também um pouco com $n_{\mathrm{H}} = 10^{-1}$ cm^{-3} neste *redshift*.

Também investigamos modelos com metalicidades [Z/H] = -1 e [Z/H] = -2. O principal efeito ocasionado pela diminuição da metalicidade é o de aumentar a temperatura, e com isso diminuir os povoamentos dos níveis de estrutura fina, uma vez que em geral estamos no intervalo em que as taxas de excitação por elétrons caem com a temperatura. Note que nos gráficos as densidades de coluna para [Z/H] = -1 foram multiplicadas por um fator 10, para efeitos de comparação. As discrepâncias maiores nas densidades de coluna individuais N_2 se dão para o Fe^+, o que é um indicativo de que a abundância fracional deste íon é mais sensível às variações de temperatura. Os resultados para [Z/H] = -2 não foram apresentados nas figuras 9.2-9.4 pelo fato de as curvas correspondentes serem praticamente coincidentes com as de [Z/H] = -1.

O tratamento empregado do cálculo do balanceamento estatístico camada a camada é importante apenas nas densidades de coluna mais altas, características dos sistemas DLA. As discrepâncias com o modelo simplificado atingem 10% em $\log N(\mathrm{H\,I}) = 19$ (ou $\log N(\mathrm{H\,I}) = 18.5$ em alguns casos), aumentando para densidades de coluna maiores até chegar a cerca de 20-65% (conforme a densidade n_{H}) para o C^+, 10-40% para o Si^+ (270% para $n_{\mathrm{H}} = 1\ \mathrm{cm}^{-3}$) e 130-180% para o Fe^+.

A Figura 9.5 mostra as dimensões e massas das regiões absorventes. Podemos então comprovar que densidades da ordem de $n_{\mathrm{H}} = 10^{-2}\ \mathrm{cm}^{-3}$ realmente implicam em tamanhos característicos da ordem de kpc, como apontado na seção 9.1 .

9.3.2. O campo do halo quente

Passamos agora a considerar o caso em que os sistemas LL são fotoionizados pelo campo local de radiação do halo quente. Como o campo integrado dos QSOs está sempre presente e não pode ser "desligado", a sua contribuição também foi adicionada, o que de qualquer forma não altera os modelos, uma vez que o campo do halo quente é muito mais intenso (Figura 9.1).

Os resultados para o campo do halo quente são mais interessantes do que aqueles obtidos para o campo integrado dos QSOs, pois, nos modelos que vamos apresentar, as variações de temperatura são maiores e o campo de radiação é intenso o bastante para tornar a fluorescência um mecanismo de excitação competitivo com colisões com elétrons.

As figuras 9.6-9.8 mostram as densidades de coluna dos níveis de estrutura fina do C^+, Si^+ e Fe^+ como função da densidade de coluna de hidrogênio neutro. Pudemos constatar que as razões de densidades de coluna dos níveis de estrutura fina são, em geral, maiores do que as obtidas nos modelos ionizados unicamente pelo campo integrado dos QSOs, se comparadas sob as mesmas condições físicas. A atuação do campo de radiação do halo quente para que isto ocorra se dá sob dois aspectos: aumentando a contribuição da fluorescência como mecanismo de excitação, e ainda tornando o meio mais ionizado, com uma densidade eletrônica maior.

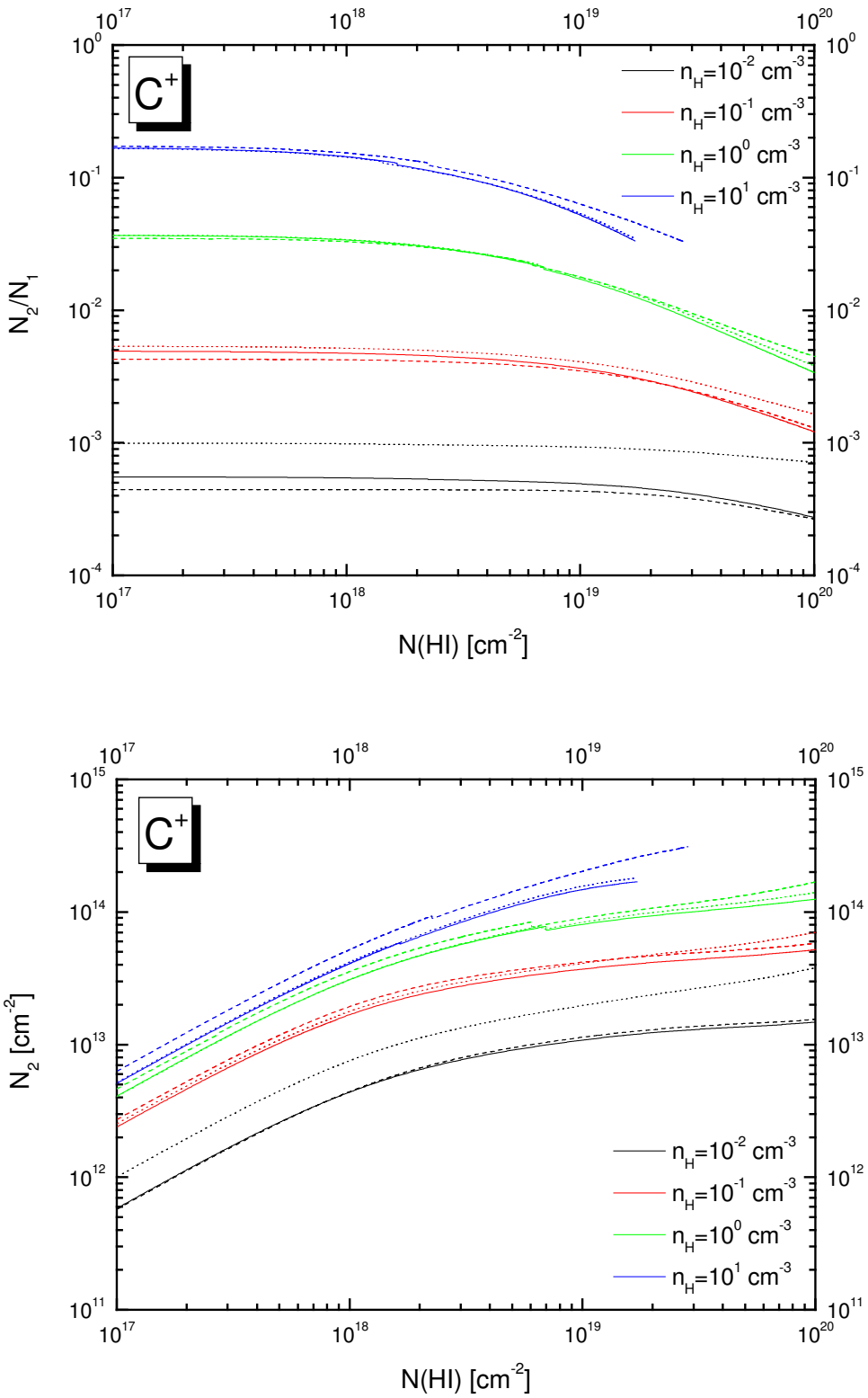


Figura 9.2: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do C^+ (QSOs de fundo).

Razões de povoamento (acima) e densidades de coluna (abaixo) dos níveis de estrutura fina do C^+ em função da densidade de coluna de hidrogênio neutro N(HI) para inúmeros valores de densidade total de hidrogênio n_H. Os modelos calculados empregam o campo integrado dos QSOs, abundâncias solares e z = 2. As curvas tracejadas indicam o efeito produzido ao se alterar a metalicidade para 10% da solar, ao passo que as curvas pontilhadas se referem a z = 3. No painel de baixo as densidades de coluna para [Z/H] = -1 foram multiplicadas por 10, para efeitos de comparação.

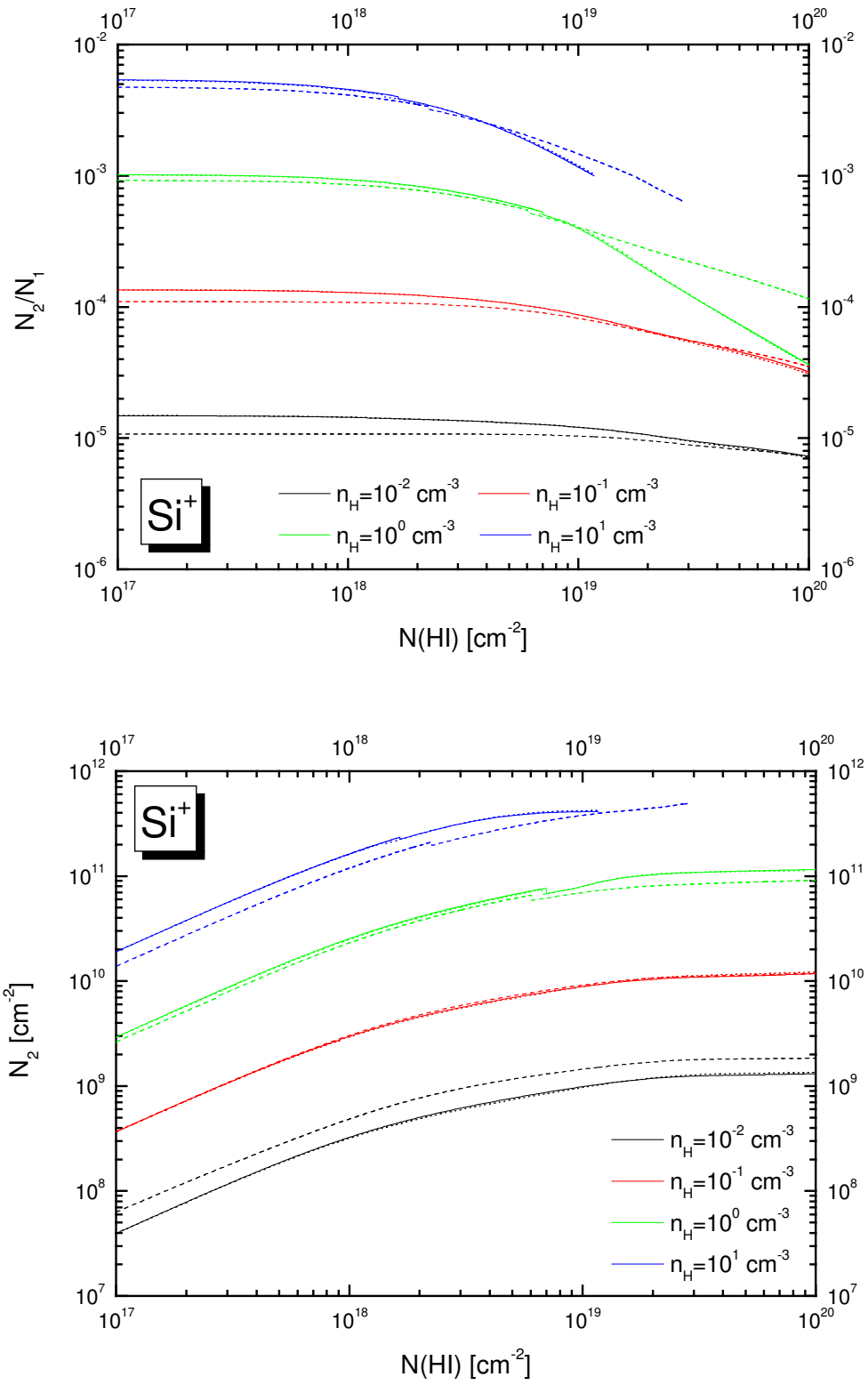


Figura 9.3: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do Si^+ (QSOs de fundo).

Razões de povoamento (acima) e densidades de coluna (abaixo) dos níveis de estrutura fina do Si^+ em função da densidade de coluna de hidrogênio neutro N(HI) para inúmeros valores de densidade total de hidrogênio n_H. Os modelos calculados empregam o campo integrado dos QSOs, abundâncias solares e z = 2. As curvas tracejadas indicam o efeito produzido ao se alterar a metalicidade para 10% da solar, ao passo que as curvas pontilhadas se referem a z = 3. No painel de baixo as densidades de coluna para [Z/H] = -1 foram multiplicadas por 10, para efeitos de comparação.

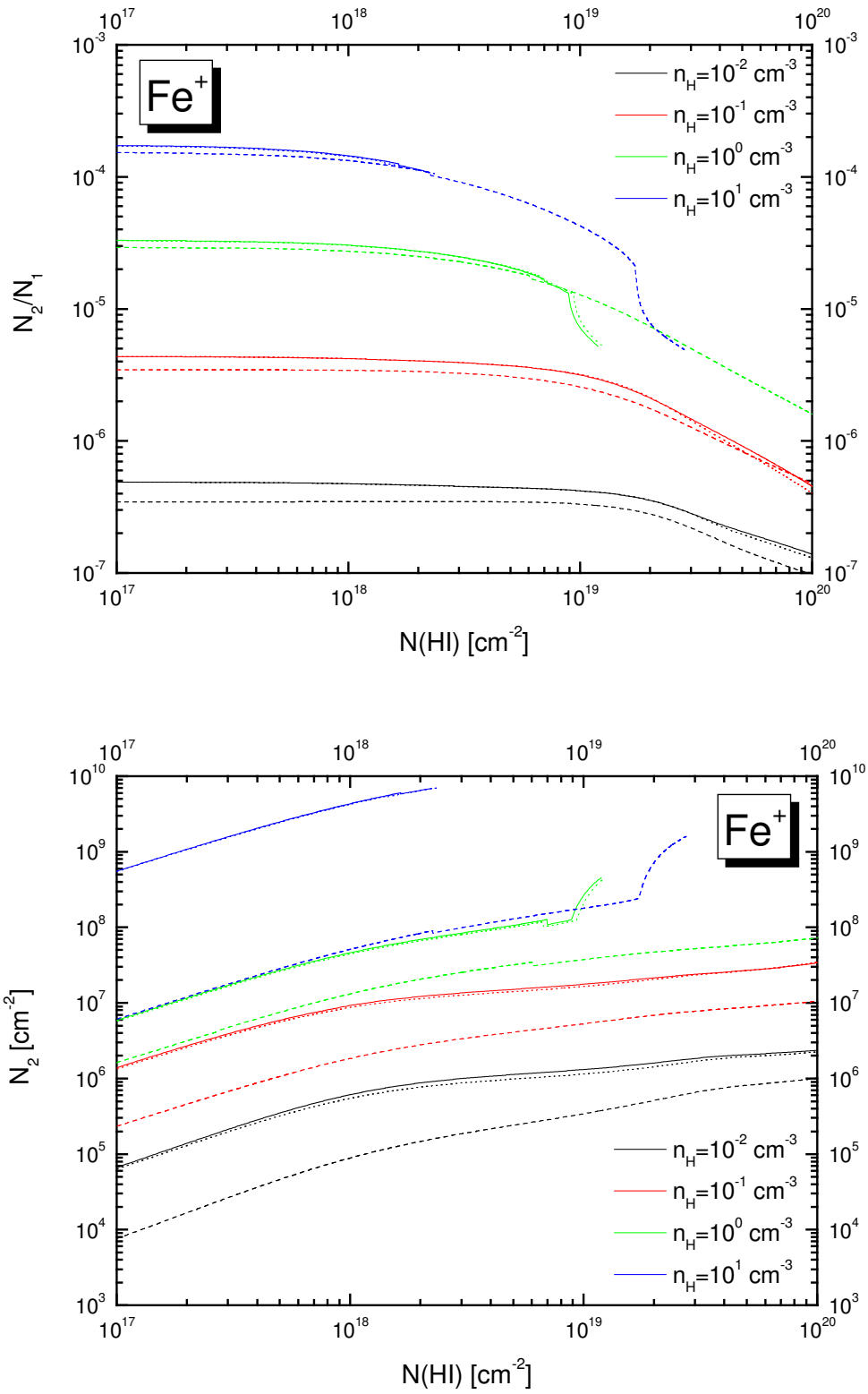


Figura 9.4: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do Fe^+ (QSOs de fundo).

Razões de povoamento (acima) e densidades de coluna (abaixo) dos níveis de estrutura fina do Fe^+ em função da densidade de coluna de hidrogênio neutro N(HI) para inúmeros valores de densidade total de hidrogênio n_H. Os modelos calculados empregam o campo integrado dos QSOs, abundâncias solares e z = 2. As curvas tracejadas indicam o efeito produzido ao se alterar a metalicidade para 10% da solar, ao passo que as curvas pontilhadas se referem a z = 3. No painel de baixo as densidades de coluna para [Z/H] = -1 foram multiplicadas por 10, para efeitos de comparação.

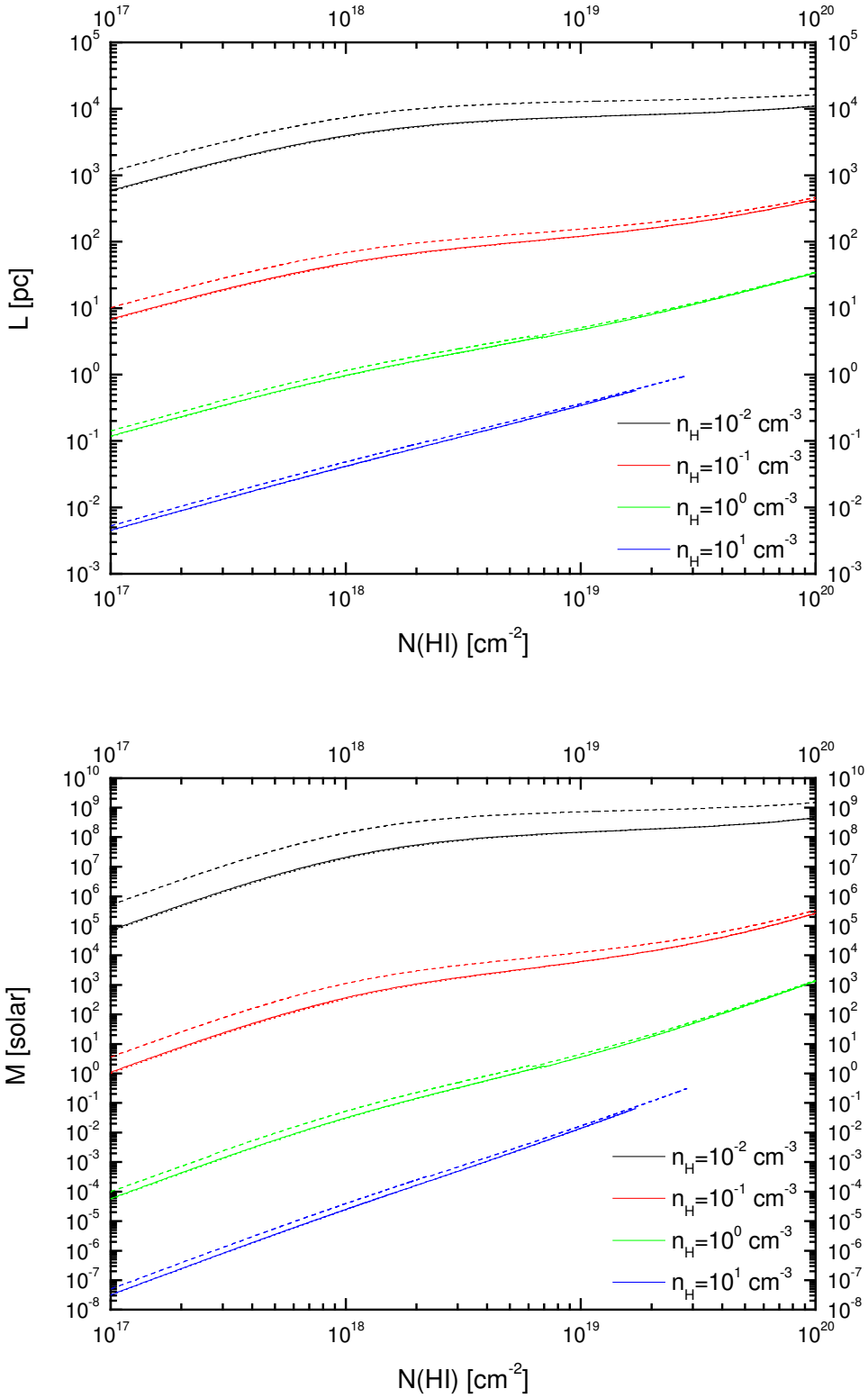


Figura 9.5: Tamanhos característicos e massas das regiões absorventes (QSOs de fundo).

Tamanhos característicos (acima) e massas (abaixo) das regiões absorventes em função da densidade de coluna de hidrogênio neutro N(HI) para inúmeros valores de densidade total de hidrogênio n_H. Os modelos calculados empregam o campo integrado dos QSOs, abundâncias solares e z = 2. As curvas tracejadas indicam o efeito produzido ao se alterar a metalicidade para 10% da solar, ao passo que as curvas pontilhadas se referem a z = 3.

Consideremos inicialmente as razões de densidades de coluna do C^+ (Figura 9.6, acima). Na curva correspondente a $n_H = 10^{-2}$ cm^{-3}, temos que colisões com elétrons e fluorescência são mecanismos de excitação igualmente importantes no início. À medida que avançamos para densidades de coluna mais altas, as colisões com elétrons tornam-se cada vez mais importantes devido à queda da temperatura, ao passo que a fluorescência cai. Nas outras curvas, colisões com elétrons são o único mecanismo de excitação relevante, e as variações nas razões de densidades de coluna são causadas unicamente pelas mudanças de temperatura. Apenas nas densidades de coluna mais altas com $n_H = 1$ cm^{-3} e $n_H = 10$ cm^{-3} as colisões com átomos de hidrogênio chegam a desempenhar algum papel.

Vamos agora concentrar nossa atenção nas razões de densidades de coluna dos níveis de estrutura fina do Si^+ (Figura 9.7, acima). Na curva correspondente a $n_H = 10^{-2}$ cm^{-3} a fluorescência sempre domina, e a taxa de excitação varia apenas lentamente no início; a taxa de excitação por colisões com elétrons chega a atingir apenas a metade da taxa de fluorescência. Com $n_H = 10^{-1}$ cm^{-3} a situação se inverte, e as taxas de excitação por elétrons são um fator 2 a 4 maiores do que as taxas de fluorescência. Nas outras curvas apenas colisões com elétrons são importantes, sendo que as taxas de excitação caem apenas em densidades de coluna muito altas.

Por fim, vamos analisar as razões de densidades de coluna dos níveis de estrutura fina do Fe^+ (Figura 9.8, acima). Aqui temos o efeito interessante em que as taxas de fluorescência aumentam à medida que adentramos o interior da nuvem. Este efeito é causado pelo aumento de fótons do difuso nas freqüências específicas das transições ultra-violeta incluídas no modelo atômico. Com $n_H = 10^{-2}$ cm^{-3} a fluorescência é o mecanismo mais importante no início, com colisões com elétrons também contribuindo. Os dois mecanismos tornam-se mais intensos, porém a fluorescência cresce bem mais rapidamente até se tornar o único mecanismo relevante. Com $n_H = 10^{-1}$ cm^{-3} as colisões com elétrons são mais importantes em densidades de coluna baixas, porém a fluorescência torna-se mais importante mais adiante. Com $n_H = 1$ cm^{-3} as taxas de excitação por colisões com elétrons são muito mais altas que as taxas de fluorescência, porém a situação se inverte nas densidades de coluna mais altas devido a uma queda brusca da densidade eletrônica. Com $n_H = 10$ cm^{-3} as taxas de excitação por colisões com elétrons são extremamente altas de modo que a fluorescência nunca consegue competir.

Ao passarmos de $z = 2$ para $z = 3$, o único efeito perceptível produzido nos modelos é o de aumentar as razões de densidades de coluna do C^+ para $n_H = 10^{-2}$ cm^{-3} e $n_H = 10^{-1}$ cm^{-3}, e esta é a única situação em que a radiação cósmica de fundo desempenha algum papel. Como o campo do

halo quente não depende do *redshift*, teremos então que todos os outros resultados aqui apresentados serão válidos também para qualquer $z \leq 3$.

A exemplo do que aconteceu com os modelos com o campo integrado dos QSOs, pudemos verificar que ao reduzirmos a metalicidade de solar para [Z/H] = -1 os povoamentos relativos dos níveis decaem, em decorrência do aumento da temperatura. Entretanto, as mudanças nas densidades de coluna individuais são mais pronunciadas, pois a variação nas abundâncias fracionais é maior. Nos modelos com [Z/H] = -2 as temperaturas nas regiões externas da nuvem são maiores do que 40.000 K, para qual os modelos de níveis atômicos empregados não são válidos.

Como as condições físicas variam mais rapidamente nos modelos com o campo de radiação do halo quente, teremos que o tratamento do balanceamento estatístico camada a camada aqui utilizado será importante em um intervalo maior de densidades de coluna. Na maioria das vezes este intervalo abrange toda a gama de densidades de coluna investigadas, $17 \leq \log N(\mathrm{H\ I}) \leq 20$. As discrepâncias com o modelo simplificado nas razões de densidades de coluna podem chegar a atingir 74% para o C^+, 40% para o Si^+ e 86% para o Fe^+. As exceções são os modelos com $n_{\mathrm{H}} = 1\ \mathrm{cm}^{-3}$ para o C^+ e $n_{\mathrm{H}} = 10\ \mathrm{cm}^{-3}$ para o C^+ e Fe^+, onde as diferenças só são maiores do que 10% para $N(\mathrm{H\ I}) > 10^{19}\ \mathrm{cm}^{-2}$.

A Figura 9.9 mostra os tamanhos e as massas das nuvens. Podemos verificar que para $n_{\mathrm{H}} = 10\ \mathrm{cm}^{-3}$ os tamanhos característicos previstos pelos modelos com o campo do halo quente são da ordem de pc.

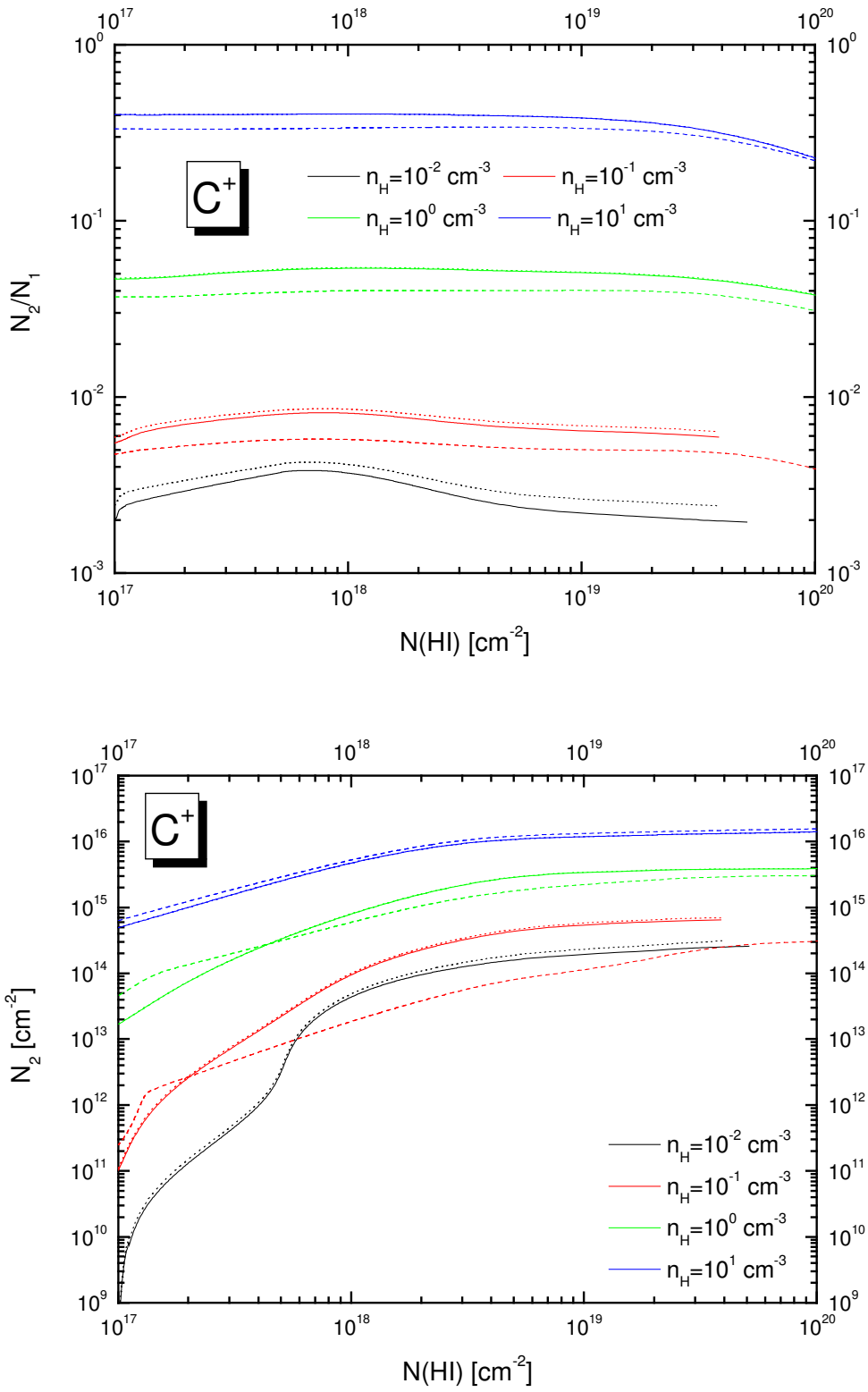


Figura 9.6: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do C^+ (halo quente).

Razões de povoamento (acima) e densidades de coluna (abaixo) dos níveis de estrutura fina do C^+ em função da densidade de coluna de hidrogênio neutro N(HI) para inúmeros valores de densidade total de hidrogênio n_H. Os modelos calculados empregam o campo do halo quente sobreposto ao campo integrado dos QSOs, abundâncias solares e z = 2. As curvas tracejadas indicam o efeito produzido ao se alterar a metalicidade para 10% da solar, ao passo que as curvas pontilhadas se referem a z = 3. No painel de baixo as densidades de coluna para [Z/H] = -1 foram multiplicadas por 10, para efeitos de comparação.

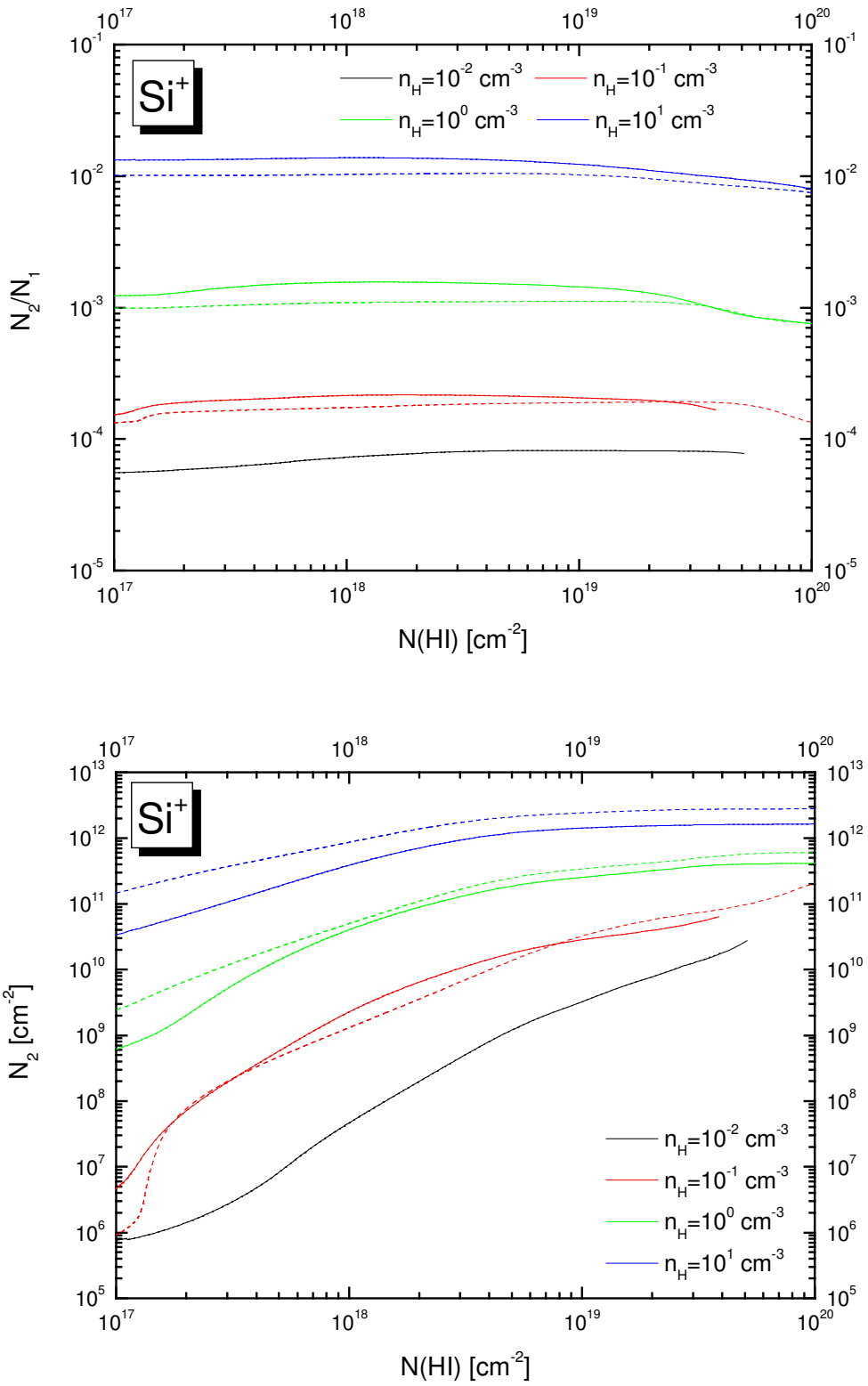


Figura 9.7: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do Si^+ (halo quente).

Razões de povoamento (acima) e densidades de coluna (abaixo) dos níveis de estrutura fina do Si^+ em função da densidade de coluna de hidrogênio neutro N(HI) para inúmeros valores de densidade total de hidrogênio n_H. Os modelos calculados empregam o campo do halo quente sobreposto ao campo integrado dos QSOs, abundâncias solares e z = 2. As curvas tracejadas indicam o efeito produzido ao se alterar a metalicidade para 10% da solar, ao passo que as curvas pontilhadas se referem a z = 3. No painel de baixo as densidades de coluna para [Z/H] = -1 foram multiplicadas por 10, para efeitos de comparação.

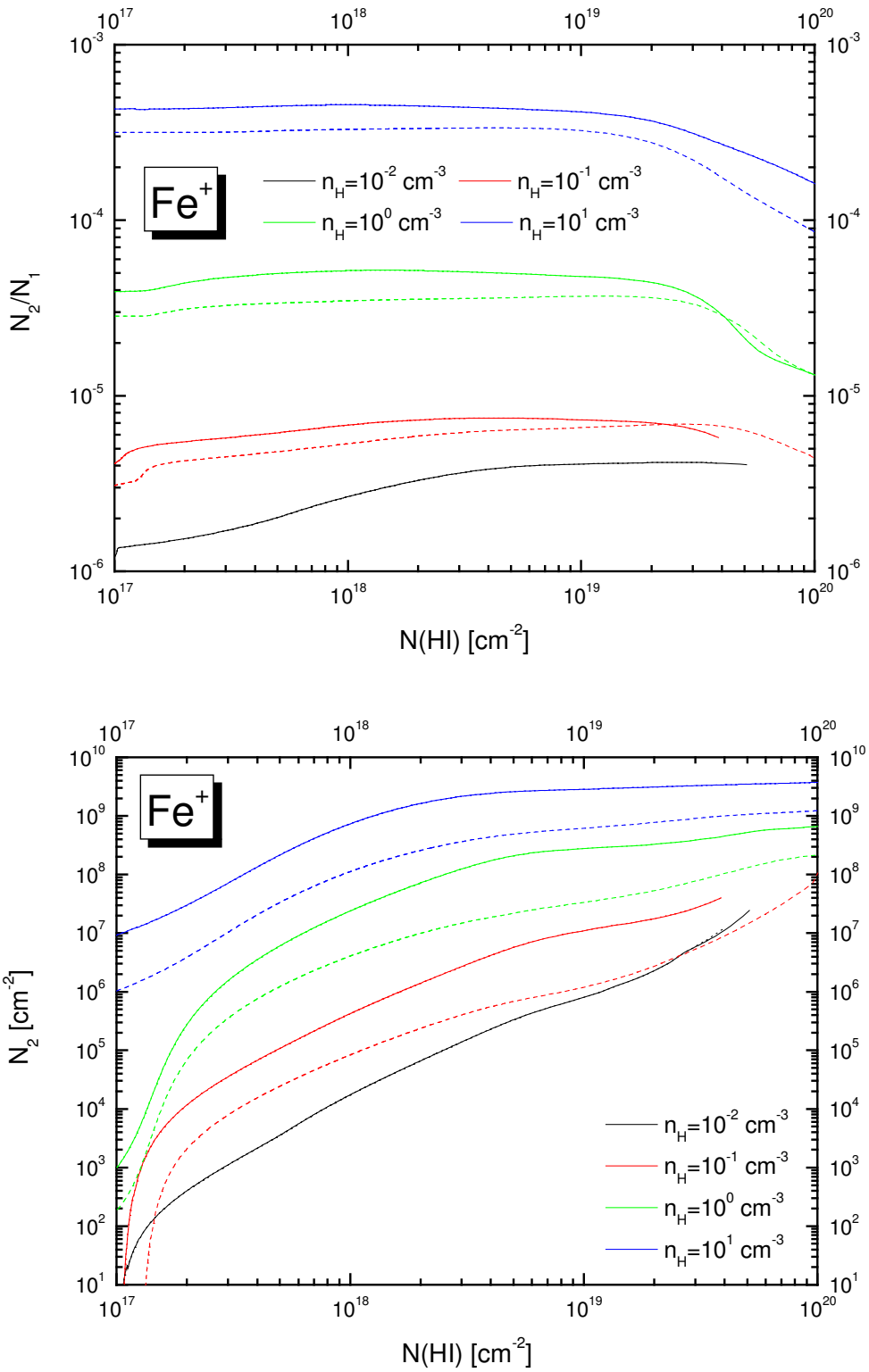


Figura 9.8: Razões de povoamento e densidades de coluna dos níveis de estrutura fina do Fe^+ (halo quente).

Razões de povoamento (acima) e densidades de coluna (abaixo) dos níveis de estrutura fina do Fe^+ em função da densidade de coluna de hidrogênio neutro N(HI) para inúmeros valores de densidade total de hidrogênio n_H. Os modelos calculados empregam o campo do halo quente sobreposto ao campo integrado dos QSOs, abundâncias solares e z = 2. As curvas tracejadas indicam o efeito produzido ao se alterar a metalicidade para 10% da solar, ao passo que as curvas pontilhadas se referem a z = 3. No painel de baixo as densidades de coluna para [Z/H] = -1 foram multiplicadas por 10, para efeitos de comparação.

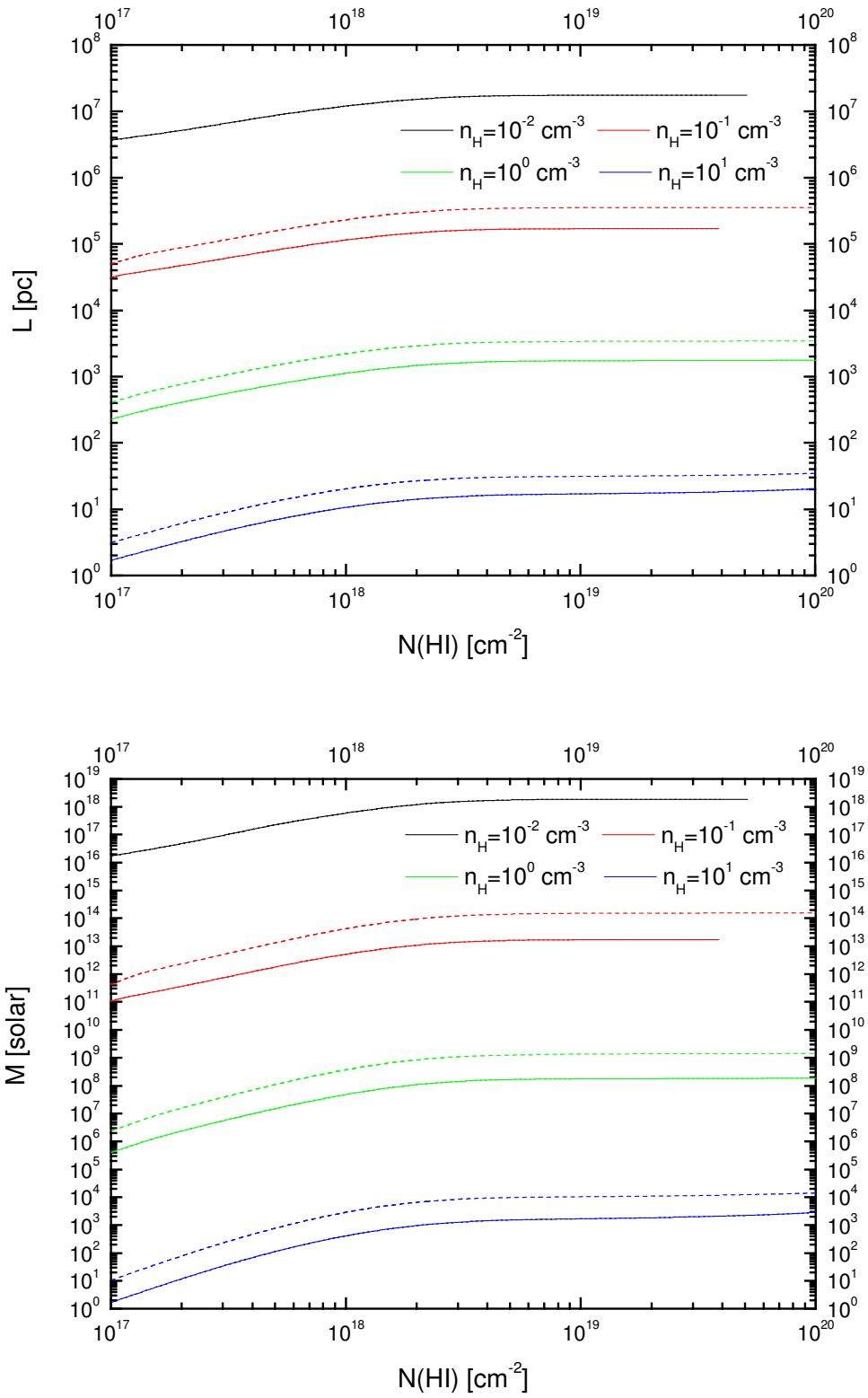


Figura 9.9: Tamanhos característicos e massas das regiões absorventes (halo quente).

Tamanhos característicos (acima) e massas (abaixo) das regiões absorventes em função da densidade de coluna de hidrogênio neutro N(HI) para inúmeros valores de densidade total de hidrogênio n_H. Os modelos calculados empregam o campo do halo quente sobreposto ao campo integrado dos QSOs, abundâncias solares e z = 2. As curvas tracejadas indicam o efeito produzido ao se alterar a metalicidade para 10% da solar, ao passo que as curvas pontilhadas se referem a z = 3.

9.4. Conclusão

Neste capítulo apresentamos resultados inéditos das razões de densidades de coluna dos níveis de estrutura fina do C^+, Si^+ e Fe^+ nos sistemas LL obtidos através de um tratamento sofisticado que considera o equilíbrio estatístico camada a camada da nuvem.

A versão eletrônica da grade de modelos produzida está disponível no CD que acompanha a tese. Além de todas as quantidades plotadas nas figuras 9.2-9.9, os arquivos de dados também contém as condições físicas em todas as camadas da nuvem juntamente com todas as taxas ligado-ligado dos processos de excitação dos níveis de estrutura fina. Apenas os campos de radiação em cada camada da nuvem não puderam ser incluídos devido ao grande volume de dados.

Note que o termo fundamental do Fe^+ é composto por cinco níveis de estrutura fina. Devido a limitações de espaço, analisamos em detalhes apenas a razão de densidades de coluna envolvendo o primeiro estado excitado e o nível fundamental. As densidades de coluna para os demais níveis de estrutura fina estão disponíveis nos arquivos de dados eletrônicos.

Modelos com condições físicas diferentes das consideradas em nossa grade poderão ser calculados sob demanda.

Algumas sugestões para refinar os cálculos ainda dentro do esquema dos modelos unidimensionais seriam:

- Aperfeiçoar o tratamento do transporte de radiação para considerar o campo que vem de ambos os lados da nuvem;

- Acoplar o balanceamento estatístico dos níveis de estrutura fina com o equilíbrio de ionização, necessário para incluir o O^0,

- Fazer o transporte das linhas, o que talvez pode ser importante em situações em que o campo é um mecanismo significativo de excitação.

Acreditamos que a questão em aberto relacionada à fonte de ionização dos sistemas LL poderá ser elucidada num futuro próximo por intermédio do estudo das linhas de estrutura fina.

Referências

Anders, E., & Grevesse, N. 1989, Geochim. Cosmochim. Acta, 53, 197

Friaça, A. C. S., & Terlevich, R. J. 1998, MNRAS, 298, 399

Gruenwald, R., & Viegas, S. M. 1993, ApJ, 415, 534

Haardt, F., & Madau, P. 1996, ApJ, 461, 20

Prochaska, J. X. 1999, ApJ, 511, L71

Reimers, D., & Vogel, S. 1993, A&A, 276, L13

Silva, A. I. 1999, dissertação de mestrado, *"Linhas de estrutura fina em absorção no espectro de QSOs."* (Instituto Astronômico e Geofísico, Universidade de São Paulo, São Paulo) http://www.ignacioalex.com/bibliografia

Silva, A. I., & Viegas, S. M. 2002, MNRAS, 329, 135

Viegas, S. M., & Friaça, A. C. S. 1995, MNRAS, 272, L35

Vogel, S., & Reimers, D. 1993, A&A, 274, L5

10. Conclusões e Perspectivas

Este trabalho tem um caráter multidisciplinar abrangendo os sistemas em absorção dos QSOs, a revisão de dados atômicos utilizados em simulações numéricas de nebulosas ionizadas, e a utilização da programação orientada a objetos em aplicações científicas.

Inicialmente calculamos as razões de povoamento dos níveis de estrutura fina do O^0 e Fe^+, complementando o trabalho realizado em Silva (1999) para o C^0, C^+ e Si^+. Com isso, completamos o estudo dos cinco íons de interesse para os sistemas em absorção dos QSOs (Silva & Viegas 2002), por estes serem os únicos íons a possuírem linhas ressonantes com comprimentos de onda superiores à transição de Ly-α do hidrogênio.

Nesta etapa também desenvolvemos um código em Fortran para resolver as equações de equilíbrio estatístico de níveis atômicos. Este novo código é muito mais flexível do que aquele empregado em Silva (1999), e pode ser empregado não apenas nas linhas em absorção, mas também no cálculo de razões de emissividades de linhas excitadas colisionalmente e de funções de resfriamento radiativo (Silva & Viegas 2001).

O objetivo final deste trabalho é ter um instrumento de análise detalhada dos sistemas em absorção. Para isso foi preciso incorporar o programa acima a um código de fotoionização. O código utilizado foi o Aangaba, desenvolvido pelo grupo, para o qual foi necessário uma revisão dos dados atômicos face aos resultados recentes nesta área. A atualização do Aangaba compreendeu as taxas de troca de carga com hidrogênio e hélio e ionização colisional. Infelizmente, constatamos que não existe na literatura um conjunto completo e auto-consistente de coeficientes de recombinação e seções de choque de fotoionização para todos os elementos do H ao Zn. Mesmo para os íons dos elementos mais abundantes, cujos coeficientes de recombinação foram calculados pelo grupo de Ohio, descobrimos problemas relacionados à resolução das ressonâncias nas seções de choque. Estes resultados servirão de estímulo para o cálculo de uma nova geração de dados mais confiáveis pelo método da matriz R, empregando esquemas como aquele proposto por Ramírez & Bautista (2002). Acreditamos que esta situação deve melhorar com o surgimento de novos dados experimentais de qualidade suficiente para calibrar os resultados teóricos.

Com o intuito de facilitar futuras atualizações como esta, elaboramos para o banco de dados atômicos (BDA) uma estrutura simples e flexível que permite a incorporação de novos dados que continuarão surgindo na literatura, sem muito esforço e com um risco mínimo de introdução de

erros. Outro ponto importante é que o BDA foi concebido para ser utilizado por outros autores em outras aplicações de interesse astrofísico.

Para projetar a implementação do BDA em *software* lançamos mão de ferramentas modernas de programação: programação orientada a objetos (POO) e UML. Estas ferramentas são ideais para lidar com a complexidade de códigos científicos muito grandes. Acreditamos que a tendência natural seja utilizá-las cada vez mais, tornando a física computacional uma vertente mais especializada, ao lado da física teórica e experimental. Dentro deste cenário, a etapa seguinte deste trabalho será estender o projeto do BDA para abranger todo um código de fotoionização. Conforme apresentado no capítulo 8, este novo código de fotoionização teria uma estrutura modular, cujos blocos individuais poderiam ser reutilizados por outros autores em suas aplicações. Com isto estaríamos usufruindo de mais um dos inúmeros benefícios decorrentes da POO, que é justamente o de promover o reuso de código.

A grande maioria dos códigos numéricos utilizados em física e astronomia empregam a linguagem de programação Fortran. Infelizmente esta linguagem não é adequada à POO, e provavelmente não o será nem mesmo após a adoção do novo padrão da linguagem. Através de nossa experiência ao implementar o BDA em Fortran, constatamos algumas falhas no esquema proposto para emular POO em Fortran 90/95 que escaparam à atenção dos outros autores, e também algumas omissões importantes no Fortran 2003. Com isto, aqueles que desejarem continuar usufruindo das facilidades numéricas e da expressividade do Fortran para aplicações científicas extremamente complexas, terão necessariamente de recorrer a mais de uma linguagem de programação.

A etapa final deste trabalho consistiu em integrar os códigos de povoamento estatístico dos níveis e também o código do BDA com o de fotoionização Aangaba, para aplicá-los aos sistemas em absorção dos QSOs, em particular os Lyman-Limit (LL). A análise detalhada destes sistemas deve complementar o conhecimento adquirido através dos sistemas Damped Lyman-α, já bastante estudados. Os sistemas LL constituem um terreno fértil a ser explorado com as ferramentas fornecidas neste trabalho. Uma questão interessante em aberto diz respeito à fonte de ionização dos LL, a qual poderá encontrar respostas nas futuras observações em telescópios de grande porte. Linhas de íons em diferentes estágios de ionização servem para restringir o parâmetro de ionização do gás e a forma do espectro de radiação ionizante, enquanto que as linhas de estrutura fina fornecem um limite independente ao parâmetro de ionização através da fixação da densidade. Como presentemente não existem medidas das linhas de estrutura fina nos sistemas LL, construímos uma grade de modelos para a análise de futuras observações.

Enfim, trilhamos um caminho longo e árduo por várias áreas, colhendo frutos nem sempre de fácil acesso em uma jornada interdisciplinar.

Referências

Ramírez, J. M., & Bautista, M. A. 2002, J. Phys. B, 35, 4139

Silva, A. I. 1999, dissertação de mestrado, *"Linhas de estrutura fina em absorção no espectro de QSOs."* (Instituto Astronômico e Geofísico, Universidade de São Paulo, São Paulo) http://www.ignacioalex.com/bibliografia

Silva, A. I., & Viegas, S. M. 2001, Comp. Phys. Comm., 136, 319

---. 2002, MNRAS, 329, 135

A. BDA: Ajustes e Exemplo

Notas:

1) Na tabela abaixo $E_1(x)$, $f_1(x)$ e $f_2(x)$ são funções especiais, vistas em detalhes no apêndice B;

2) K2eV denota a constante de conversão de K para eV, de modo que se multiplicarmos uma quantidade *x* expressa em K por este número, obteremos a quantidade correspondente *x'* expressa em eV (e analogamente para as demais constantes de conversão).

número do ajuste	fórmula
1	$f(x)=0$
2	$f(x)=c_1$
3	$f(x)=c_1\left(\frac{x}{10^4}\right)^{c_2}(1+c_3\ e^{c_4\frac{x}{10^4}})$
4	$f(x)=c_1\left(\frac{x}{10^4}\right)^{c_2}(1+c_3\ e^{c_4\frac{x}{10^4}})\ e^{-c_5\frac{10^4}{x}}$
5	$f(x)=c_1\left(\frac{x}{10^4}\right)^{c_2}e^{-c_3x/10^4}e^{-c_4/x}$
6	$f(x)=c_2\frac{1}{c_3+y}y^{c_4}e^{-y}$ $y=c_1/x'$ $x'=\text{K2eV}\ x$
7	$f(x)=c_2\frac{1+\sqrt{y}}{c_3+y}y^{c_4}e^{-y}$ $y=c_1/x'$ $x'=\text{K2eV}\ x$
8	$f(x)=e^{-\frac{c_1}{x'}}\sqrt{\frac{x'}{c_1}}\sum_{n=2}^{7}c_n(\log\frac{x'}{c_1})^{n-2}$ $x'=\text{K2eV}\ x$

9	$f(x)=\sqrt{\frac{c_1}{x'}}\left[c_2\ln\frac{x'}{c_1}+\sum_{n=3}^{5}c_n\left(\frac{c_1}{x'}\right)^{n-3}\right]$ $x'=\text{K2eV}\,x$
10	$f(x)=\frac{6.69\times10^{-7}}{x'^{3/2}}\frac{e^{-y}}{y}F(y)$ $y=\frac{c_1}{x'}$ $F(y)=c_2[1-yf_1(y)]+c_3[1+y-y(2+y)f_1(y)]+c_4f_1(y)+c_5yf_2(y)$ $x'=\text{K2eV}\,x$
11	$\begin{cases} f'(x)=0 & x'<c_1 \\ f'(x)=c_3(y-1)^2y^{0.5c_5-5.5}(1+\sqrt{y/c_4})^{-c_5} & x'\geq c_1 \end{cases}$ $y=x'/c_2$ $x'=\text{erg2eV}\,x$ $f=\text{Mb2cm2}\,f'$
12	$\begin{cases} f'(x)=0 & x'<c_1 \\ f'(x)=c_3[(y-1)^2+c_7^2]y^{0.5c_6-c_4-5.5}(1+\sqrt{y/c_5})^{-c_6} & x'\geq c_1 \end{cases}$ $y=x'/c_2$ $x'=\text{erg2eV}\,x$ $f=\text{Mb2cm2}\,f'$
13	$\begin{cases} f'(x)=0 & x'<c_1 \\ f'(x)=c_3[(y-1)^2+c_6^2]z^{0.5c_5-5.5}(1+\sqrt{z/c_4})^{-c_5} & x'\geq c_1 \end{cases}$ $y=x'/c_2-c_7$ $z=\sqrt{y^2+c_8^2}$ $x'=\text{erg2eV}\,x$ $f=\text{Mb2cm2}\,f'$
14	$f(x)=\frac{c_2}{\sqrt{x'}}e^{-y}\{0.67+2.22f_1(y)-0.18yf_1(y)+1.2y[1-yf_1(y)]\}$ $y=\frac{c_1}{x'}$ $x'=\text{K2eV}\,x$
15	$f(x)=\frac{c_2}{\sqrt{x'}}e^{-y}[1-yf_1(y)]$ $y=\frac{c_1}{x'}$ $x'=\text{K2eV}\,x$

16	$f(x)=\frac{c_2}{\sqrt{x'}}e^{-y}\{1-0.5[y-y^2+y^3 f_1(y)]\}$ $y=\frac{c_1}{x'}$ $x'=\text{K2eV}\ x$
17	$f(x)=\frac{c_2}{\sqrt{x'}}e^{-y}[1+1.12 f_1(y)]$ $y=\frac{c_1}{x'}$ $x'=\text{K2eV}\ x$
18	$f(x)=\frac{6.69\times10^{-9}}{\sqrt{x'}}e^{-y}F(y)$ $F(y)=c_2+c_3[1-yf_1(y)]+c_4\{1-y[1-yf_1(y)]\}+$ $+c_5\{1-0.5[y-y^2+y^3 f_1(y)]\}+c_6 f_1(y)$ $y=\frac{c_1}{x'}$ $x'=\text{K2eV}\ x$
19	$f(x)=c_1\left(\frac{x}{10^4}\right)^{-c_2}$
20	$f(x)=c_1\left(\frac{x}{10^4}\right)^{-c_2-c_3\log(x/10^4)}$
21	$f(x)=c_1\left[\sqrt{\frac{x}{c_3}}\left(1+\sqrt{\frac{x}{c_3}}\right)^{1-c_2}\left(1+\sqrt{\frac{x}{c_4}}\right)^{1+c_2}\right]^{-1}$
22	$f(x)=10^{-13}c_1\frac{c_2 y^{c_3}}{1+c_4 y^{c_5}}$ $y=10^{-4}x/c_1^2$
23	$f(x)=\frac{6.69\times10^{-7}}{x'^{3/2}}c_2c_3\left[\frac{E_1(y)}{y}-\frac{c_4 e^{c_5}}{y+c_5}E_1(y+c_5)\right]$ $y=\frac{c_1}{x'}$ $x'=\text{K2eV}\ x$
24	$f(x)=\frac{6.69\times10^{-7}}{x'^{3/2}}c_2c_3\frac{x'}{c_1}E_1(\frac{c_1}{x'})$ $x'=\text{K2eV}\ x$

Rates for charge transfer ionization reactions with hydrogen:

$$X^{+i} + H^{+} \rightarrow X^{+(i+1)} + H^{0}$$

Revision history (DD/MM/YYYY):

09/04/2003 original version Alex Ignácio da Silva - alex@ignacioalex.com

Contents:

Neutrals : Li+0 C+0 N+0 O+0 Na+0 Mg+0 Si+0 S+0 Fe+0 .
Singly-ionized : Mg+1 Si+1 Cr+1 Mn+1 Fe+1 Co+1 .

number of data blocks in this file:
[DATA_BLOCK]
4

Reference:

J.B. Kingdon and G.J. Ferland (1996), ApJS, 106, 205

with revisions to Li^{+0}, C^{+0}, and additions of Na^{+0} and Si^{+0} from the Charge TransferDatabase at ORNL/UGA :

Li^{+0}: original data have been revised;
C^{+0} : fit now is based on the more recent calculation of Stancil et al. (1998), ApJ, 502, 1006;
Na^{+0}: new fit based on Croft and Dickinson (1996), J. Phys. B, 29, 57,
Si^{+0}: new fit based on Kimura et al. (1996), ApJ, 473, 1114.

Fit formula #3:

$$f(x) = c_1 \left(\frac{x}{10^4} \right)^{c_2} (1 + c_3 \, e^{c_4 \frac{x}{10^4}})$$

TOP

next

number_of_entries_in_this_block
table/fit [ROW/column/multicolumn/commoncolumn/commonrow] ...
... [LINEAR_LINEAR/linear_log/.../log_/_linear/...] ... (if table)
... [eV_/eV_Mb/...] ... (if table)
... [Z_ne/Z_st][shell]
number_of_fit_formula number_of_parameters (if fit)

X+i Tmin Tmax c1 c2 c3 c4

[DATA_BLOCK]

```
5
fit
3      4
Li+0    1.0E+02    1.0E+04    2.84E-12    1.99      375.54      -54.07
C+0     1.0E+03    1.0E+05    1.07E-15    3.15      176.43       -4.29
Na+0    1.0E+03    2.0E+04    3.34E-15    9.31     2632.31       -3.04
Mg+0    5.0E+03    3.0E+04    9.76E-12    3.14       55.54       -1.12
Si+0    1.0E+03    2.0E+05    9.2E-10     1.15        0.80       -0.24
```

Reference:

J.B. Kingdon and G.J. Ferland (1996), ApJS, 106, 205

with revisions to Si^{+1} from the Charge Transfer Database at ORNL/UGA :

Si^{+1}: fit now is based on the more recent calculation of
Clarke et al. (1998), J. Phys. B, 31, 533.

Fit formula #4:

$$f(x) = c_1 \left(\frac{x}{10^4}\right)^{c_2} (1 + c_3\, e^{c_4 \frac{x}{10^4}})\; e^{-c_5 \frac{10^4}{x}}$$

previous TOP next

number_of_entries_in_this_block
table/fit [ROW/column/multicolumn/commoncolumn/commonrow] ...
... [LINEAR_LINEAR/linear_log/.../log_/_linear/...] ... (if table)
... [eV_/eV_Mb/...] ... (if table)
... [Z_ne/Z_st][shell]
number_of_fit_formula number_of_parameters (if fit)

X+i Tmin Tmax c1 c2 c3 c4 c5

```
[DATA_BLOCK]
7
fit
4      5
N+0     1.0E+02    5.0E+04    4.55E-12   -0.29      -0.92      -8.38    1.086
Mg+1    1.0E+04    3.0E+05    7.60E-14    0.00      -1.97      -4.32    1.670
Si+1    2.0E+03    1.0E+05    2.26E-09    7.36E-02  -0.43      -0.11    3.031
Cr+1    1.0E+03    3.0E+04    4.39E-09    0.61      -0.89      -3.56    3.349
Mn+1    1.0E+03    3.0E+04    2.83E-10    6.80E-03   6.44E-02  -9.70    2.368
Fe+1    1.0E+04    1.0E+05    2.10E-09    7.72E-02  -0.41      -7.31    3.005
Co+1    1.0E+03    3.0E+04    1.20E-11    3.49      24.41      -1.26    4.004
```

Reference:

J.B. Kingdon and G.J. Ferland (1996), ApJS, 106, 205

The entry for Fe^{+0} corresponds to 4/5 of the Langevin rate, as revised in the Charge Transfer Database at ORNL/UGA.

Fit formula #2:

$$f(x) = c_1$$

previous TOP next

```
number_of_entries_in_this_block
table/fit   [ROW/column/multicolumn/commoncolumn/commonrow] ...
   ... [LINEAR_LINEAR/linear_log/.../log_/_linear/...] ...  (if table)
   ... [eV_/eV_Mb/...] ...  (if table)
   ... [Z_ne/Z_st][shell]
number_of_fit_formula     number_of_parameters     (if fit)

X+i    Tmin       Tmax       c1

[DATA_BLOCK]
2
fit
2     1
S+0    1.0E+03    1.0E+04    1.00E-14
Fe+0   1.0E+01    1.0E+04    5.40E-09
```

$O(^3P_2) + H^+ \rightarrow O^+(^4S) + H$

Reference:

P.C. Stancil, D.R. Schultz, M. Kimura, J.-P. Gu, G. Hirsch and R.J. Buenker (1999), A&AS, 140, 225

Note: The point at T = 10,000 K corresponds to an average of the fine-structure resolved and the high-T values (tables 4 and 5 of the above paper).

previous TOP

```
number_of_entries_in_this_block
table/fit   [ROW/column/multicolumn/commoncolumn/commonrow] ...
   ... [LINEAR_LINEAR/linear_log/.../log_/_linear/...] ...  (if table)
   ... [eV_/eV_Mb/...] ...  (if table)
   ... [Z_ne/Z_st][shell]
number_of_fit_formula     number_of_parameters     (if fit)

T[K]    delta [cm^3 s^-1]
```

```
[DATA_BLOCK]
1
table  column
23
O0
    10.0   3.17E-20
    20.0   3.59E-15
    30.0   1.70E-13
    50.0   3.70E-12
   100.0   4.25E-11
   200.0   1.59E-10
   500.0   4.06E-10
  1000.0   6.37E-10
  2000.0   8.97E-10
  5000.0   1.30E-09
 10000.0   1.68E-09
 20000.0   2.19E-09
 50000.0   3.37E-09
 80000.0   4.22E-09
100000.0   4.71E-09
200000.0   6.59E-09
500000.0   1.03E-08
800000.0   1.29E-08
1.0E+06    1.44E-08
2.0E+06    2.02E-08
5.0E+06    3.11E-08
8.0E+06    3.86E-08
1.0E+07    4.27E-08
```

THE END

B. Algumas Funções Especiais

B1. $f_1(x) = e^x \int_1^\infty \frac{e^{-xt}}{t} dt$

Esta função pode ser calculada por intermédio das integrais exponenciais:

$$E_n = \int_1^\infty e^{-xt} t^{-n} dt \qquad n = 0, 1, 2... \ , \tag{B.1}$$

as quais ocorrem freqüentemente, e por isso já foram desenvolvidas inúmeras fórmulas simples para calculá-las rápida e eficientemente (Gautschi & Cahill 1970).

Sua expansão em série de potências resulta em:

$$f_1(x) = e^x \left[-\gamma - \ln(x) - \sum_{n=1}^{\infty} \frac{(-1)^n x^n}{nn!} \right] , \tag{B.2}$$

onde $\gamma = 0.5772\ldots$ é a constante de Euler-Mascheroni.

Para valores grandes de x é mais conveniente empregar a expansão assimptótica:

$$f_1(x) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}(n-1)!}{x^n} \ . \tag{B.3}$$

Podemos avaliar $f_1(x)$ com precisão razoável truncando as séries acima nos seguintes termos (Seaton 1959):

$$\begin{aligned} f_1(x) &\cong e^x[-\ln(x) - \gamma + x] && x \le 0.02 \\ f_1(x) &\cong \frac{1}{x}(1 - \frac{1!}{x} + \frac{2!}{x^2} - \frac{3!}{x^3} + \frac{4!}{x^4}) && x \ge 10 \end{aligned} \ . \tag{B.4}$$

Para valores intermediários de x podemos empregar a seguinte relação (Mewe & Schrijver 1978):

$$f_1(x) \cong \ln\left(\frac{x+1}{x}\right) - \frac{[0.36 + 0.03(x+0.01)^a]}{(x+1)^2} \qquad 0.02 < x < 10 \ , \tag{B.5}$$

onde $a = 0.5$ para $x \geq 1$, e $a = -0.5$ para $x < 1$.

Note que esta última equação aparece colocada incorretamente em Arnaud & Rothenflug (1985).

Outra relação conveniente para avaliar esta função é dada pela fração continuada (Gautschi & Cahill 1970):

$$f_1(x) = \frac{1}{x+}\ \frac{n}{1+}\ \frac{1}{x+}\ \frac{n+1}{1+}\ \frac{2}{x+}\ldots \ , \tag{B.6}$$

que converge para todo $x > 0$.

B2. $f_2(x) = e^x \int_1^\infty \frac{\ln(t)}{t} e^{-xt} dt$

Esta função também ocorre com uma certa freqüência; uma generalização desta foi estudada em detalhes por Hummer (1983).

A expansão em série é dada por:

$$f_2(x) = e^x \left[\frac{1}{2}(\ln x + \gamma)^2 + \frac{\pi^2}{12} + \sum_{n=1}^{\infty} \frac{(-1)^n x^n}{n^2 n!} \right] , \tag{B.7}$$

ao passo que a expansão assimptótica fornece:

$$f_2(x) = \frac{1}{x^2} \sum_{n=0}^{\infty} D_{n+1}\, x^{-n} \ , \tag{B.8}$$

onde $D_0 = 0$ e os outros coeficientes podem ser obtidos recursivamente:

$$D_{n+1} = (-1)^n n! - (n+1)D_n \ . \tag{B.9}$$

Uma maneira de se diminuir o menor valor de x para o qual a expansão assimptótica resulta numa dada precisão é expressá-la numa expressão racional de potências de $1/x$. Assim, é conveniente escrever $f_2(x)$ da seguinte forma (Hummer 1983):

$$f_2(x) = \frac{1}{x^2}\frac{P(x)}{Q(x)} \ , \tag{B.10}$$

onde os polinômios são dados por:

$$\begin{aligned} P(x) &= \sum_{n=0}^{\infty} p_n \, x^{-n} \\ Q(x) &= \sum_{n=0}^{\infty} q_n \, x^{-n} \end{aligned} \ , \tag{B.11}$$

e os coeficientes p_n e q_n são dados na tabela (B.1).

Para valores pequenos de x Verner (ADfA) aproxima $f_2(x)$ pelos termos de mais baixa ordem em (B.7):

$$f_2(x) \cong \frac{1}{2}(\ln x + \gamma)^2 + \frac{\pi^2}{12} \qquad x < 0.27 \ . \tag{B.12}$$

n	p_n	q_n
0	1.000000000000000E+00	1.000000000000000E+00
1	2.165775917955525E+02	2.195775917955525E+02
2	2.033587105342909E+04	2.098360382881574E+04
3	1.091065065646484E+06	1.151650523623183E+06
4	3.711399580079484E+07	4.034883260913706E+07
5	8.396327345237503E+08	9.490018565225945E+08
6	1.288918149583380E+10	1.534455719230511E+10
7	1.344856883951851E+11	1.718164841585205E+11
8	9.400171941902539E+11	1.324851363645336E+12
9	4.257075535864156E+12	6.907059906460437E+12
10	1.174304576501000E+13	2.353091122612837E+13
11	1.754867504059569E+13	4.943221846942425E+13
12	1.080630148187369E+13	5.776015744186425E+13
13	4.977620100319121E+11	3.022538587683950E+13
14		3.364065905726312E+12

Tabela B.1: Coeficientes p_n e q_n da expansão racional de $f_2(x)$.

Extraído de Hummer (1983).

Referências

ADfA, Banco de dados atômicos do código de fotoionização Cloudy: http://www.pa.uky.edu/~verner/atom.html

Arnaud, M., & Rothenflug, R. 1985, A&AS, 60, 425

Gautschi, W., & Cahill, W. F. 1970, in *"Handbook of Mathematical Functions"*, eds. M. Abramowitz, & I. A. Stegun, 9ª ed. (Nova Iorque: Dover Publications, Inc.), p. 227

Hummer, D. G. 1983, J. Quant. Spectrosc. Radiat. Transfer, 30, 281

Mewe, R., & Schrijver, J. 1978, A&A, 65, 99

Seaton, M. J. 1959, MNRAS, 119, 81

C. Códigos Fonte

C1. meuPacote_Ancestral_Class.f90

```
! Esta classe implementa a interface Funcao.
module meuPacote_Ancestral_Class

implicit none

! acessibilidade padrão:
private

! atributos no escopo da classe:
integer :: Ancestral_nInstancias = 0

! necessário para que as variáveis não fiquem indefinidas:
save

type, public :: Ancestral

    ! atributos no escopo da instância:
    private
    real :: atributo1

end type Ancestral

! métodos públicos:
public :: new, finish
public :: Ancestral_getNInstancias, getAtributo1, setAtributo1
public :: avalie

! interfaces para métodos no escopo da instância:

interface new
    module procedure Ancestral_new
end interface new

interface finish
    module procedure Ancestral_finish
end interface finish

interface getAtributo1
    module procedure Ancestral_getAtributo1
end interface getAtributo1

interface setAtributo1
    module procedure Ancestral_setAtributo1
end interface setAtributo1
```

```
interface avalie
    module procedure Ancestral_avalie
end interface avalie

contains

!********************
!* métodos públicos *
!********************

! construtor:
subroutine Ancestral_new (this,valor1)
type(Ancestral), intent(inout) :: this
real, intent(in) :: valor1

call setAtributo1(this,valor1)
call Ancestral_incNInstancias()

end subroutine Ancestral_new

! destrutor:
subroutine Ancestral_finish (this)
type(Ancestral), intent(inout) :: this

call Ancestral_decNInstancias()

end subroutine Ancestral_finish

function Ancestral_getNInstancias () result(nInstancias)
integer :: nInstancias

nInstancias = Ancestral_nInstancias

end function Ancestral_getNInstancias

function Ancestral_getAtributo1 (this) result(valor1)
type(Ancestral), intent(in) :: this
real :: valor1

valor1 = this%atributo1

end function Ancestral_getAtributo1

subroutine Ancestral_setAtributo1 (this,valor1)
type(Ancestral), intent(inout) :: this
real, intent(in) :: valor1

this%atributo1 = valor1

end subroutine Ancestral_setAtributo1

function Ancestral_avalie (this,x) result(valor)
type(Ancestral), intent(in) :: this
real, intent(in) :: x
real :: valor

valor = x * this%atributo1

end function Ancestral_avalie
```

```
!********************
!* métodos privados *
!********************

subroutine Ancestral_incNInstancias ()

Ancestral_nInstancias = Ancestral_nInstancias + 1

end subroutine Ancestral_incNInstancias

subroutine Ancestral_decNInstancias ()

Ancestral_nInstancias = Ancestral_nInstancias - 1

end subroutine Ancestral_decNInstancias

end module meuPacote_Ancestral_Class
```

C2. meuPacote_Descendente_Class.f90

```
! Esta classe estende a classe Ancestral.
module meuPacote_Descendente_Class

use meuPacote_Ancestral_Class

implicit none

! acessibilidade padrão:
private

! esta classe estende Ancestral:
type, public :: Descendente

    ! atributos no escopo da instância:
    private
    type(Ancestral) :: super
    real :: atributo2

end type Descendente

! métodos públicos:
public :: new, finish
public :: Descendente_getNInstancias, getAtributo1, setAtributo1
public :: getAtributo2, setAtributo2
public :: avalie

! interfaces para métodos no escopo da instância:

interface new
    module procedure Descendente_new
end interface new

interface finish
    module procedure Descendente_finish
end interface finish

interface getAtributo1
    module procedure Descendente_getAtributo1
end interface getAtributo1

interface setAtributo1
    module procedure Descendente_setAtributo1
end interface setAtributo1

interface getAtributo2
    module procedure Descendente_getAtributo2
end interface getAtributo2

interface setAtributo2
    module procedure Descendente_setAtributo2
end interface setAtributo2

interface avalie
    module procedure Descendente_avalie
end interface avalie
```

```
contains

!********************
!* métodos públicos *
!********************

! construtor:
subroutine Descendente_new (this,valor1,valor2)
type(Descendente), intent(inout) :: this
real, intent(in) :: valor1, valor2

! chama o construtor da superclasse:
call new(this%super,valor1)

call setAtributo2(this,valor2)

end subroutine Descendente_new

! destrutor:
subroutine Descendente_finish (this)
type(Descendente), intent(inout) :: this

call finish(this%super)

end subroutine Descendente_finish

function Descendente_getNInstancias () result(nInstancias)
integer :: nInstancias

nInstancias = Ancestral_getNInstancias()

end function Descendente_getNInstancias

function Descendente_getAtributo1 (this) result(valor1)
type(Descendente), intent(in) :: this
real :: valor1

valor1 = getAtributo1(this%super)

end function Descendente_getAtributo1

subroutine Descendente_setAtributo1 (this,valor1)
type(Descendente), intent(inout) :: this
real, intent(in) :: valor1

call setAtributo1(this%super,valor1)

end subroutine Descendente_setAtributo1

function Descendente_getAtributo2 (this) result(valor2)
type(Descendente), intent(in) :: this
real :: valor2

valor2 = this%atributo2

end function Descendente_getAtributo2
```

```
subroutine Descendente_setAtributo2 (this,valor2)
type(Descendente), intent(inout) :: this
real, intent(in) :: valor2

this%atributo2 = valor2

end subroutine Descendente_setAtributo2

! sobrepõe Ancestral.avalie(x):
function Descendente_avalie (this,x) result(valor)
type(Descendente), intent(in) :: this
real, intent(in) :: x
real :: valor

real :: atributo1  ! variável temporária para armazenar o atributo de Ancestral

! é necessário uma chamada de método para acessar o atributo protegido de Ancestral!
atributo1 = getAtributo1(this%super)

valor = x * sqrt(atributo1*atributo1 + this%atributo2*this%atributo2)

end function Descendente_avalie

end module meuPacote_Descendente_Class
```

C3. meuPacote_Funcao_Interface.f90

```
! Esta interface implementa uma função de uma variável.
module meuPacote_Funcao_Interface

! classes que implementam a interface:
use meuPacote_Ancestral_Class
use meuPacote_Descendente_Class

implicit none

! acessibilidade padrão:
private

type, public :: Funcao

    ! ponteiros para classes que implementam a interface:
    private
    type(Ancestral),   pointer :: ancestral   => null()
    type(Descendente), pointer :: descendente => null()

end type Funcao

! todos os métodos são públicos:
public :: new, finish, avalie

interface new
    module procedure Funcao_newAncestral
    module procedure Funcao_newDescendente
end interface new

interface finish
    module procedure Funcao_finish
end interface finish

interface avalie
    module procedure Funcao_avalie
end interface avalie

contains

!***********
!* métodos *
!***********

! construtor para uma instância de Ancestral:
subroutine Funcao_newAncestral (this,classname,valor1)
type(Funcao), intent(inout)  :: this
character(len=*), intent(in) :: classname
real, intent(in) :: valor1

if (classname=="Ancestral") then
    allocate (this%ancestral)
    call new(this%ancestral,valor1)
else
    write (6,*) "Erro! Funcao_newAncestral: construtor errado!"
    stop
end if

end subroutine Funcao_newAncestral
```

```
! construtor para uma instância de Descendente:
subroutine Funcao_newDescendente (this,classname,valor1,valor2)
type(Funcao), intent(inout)  :: this
character(len=*), intent(in) :: classname
real, intent(in) :: valor1, valor2

if (classname=="Descendente") then
    allocate (this%descendente)
    call new(this%descendente,valor1,valor2)
else
    write (6,*) "Erro! Funcao_newDescendente: construtor errado!"
    stop
end if

end subroutine Funcao_newDescendente

! destrutor:
subroutine Funcao_finish (this)
type(Funcao), intent(inout) :: this

if (associated(this%ancestral)) then
    call finish(this%ancestral)
    deallocate (this%ancestral)
else if (associated(this%descendente)) then
    call finish(this%descendente)
    deallocate (this%descendente)
else
    write (6,*) "Erro! Funcao_finish: nenhum objeto alocado!"
    stop
end if

end subroutine Funcao_finish

! método polimórfico:
function Funcao_avalie (this,x) result(valor)
type(Funcao), intent(in) :: this
real, intent(in) :: x
real :: valor

! delega a tarefa ao objeto apropriado:
if (associated(this%ancestral)) then
    valor = avalie(this%ancestral,x)
else if (associated(this%descendente)) then
    valor = avalie(this%descendente,x)
else
    write (6,*) "Erro! Funcao_avalie: nenhum objeto alocado!"
    stop
end if

end function Funcao_avalie

end module meuPacote_Funcao_Interface
```

C4. meuPacote_ClasseAbstrata_Class.f90

```
! Esta é uma classe abstrata.
module meuPacote_ClasseAbstrata_Class

implicit none

! acessibilidade padrão:
private

type, public :: ClasseAbstrata

    ! atributos no escopo da instância:
    private
    real :: atributo

end type ClasseAbstrata

! apenas os métodos concretos são públicos:
public :: getAtributo, setAtributo, metodo1

! interfaces para métodos no escopo da instância:

interface getAtributo
    module procedure ClasseAbstrata_getAtributo
end interface getAtributo

interface setAtributo
    module procedure ClasseAbstrata_setAtributo
end interface setAtributo

interface metodo1
    module procedure ClasseAbstrata_metodo1
end interface metodo1

interface metodo2
    module procedure ClasseAbstrata_metodo2
end interface metodo2

contains

! nenhum construtor ou destrutor é fornecido

function ClasseAbstrata_getAtributo(this) result(valor)
type(ClasseAbstrata), intent(in) :: this
real :: valor

valor = this%atributo

end function ClasseAbstrata_getAtributo

subroutine ClasseAbstrata_setAtributo(this,valor)
type(ClasseAbstrata), intent(inout) :: this
real, intent(in) :: valor

this%atributo = valor

end subroutine ClasseAbstrata_setAtributo
```

```
subroutine ClasseAbstrata_metodo1(this)
type(ClasseAbstrata), intent(inout) :: this

print *, metodo2(this)

end subroutine ClasseAbstrata_metodo1

! método abstrato:
function ClasseAbstrata_metodo2(this) result(numero)
type(ClasseAbstrata), intent(in) :: this
real :: numero

numero = 0.0

end function ClasseAbstrata_metodo2

end module meuPacote_ClasseAbstrata_Class
```

C5. meuPacote_ClasseConcreta_Class.f90

```
! Esta é uma classe concreta que estende ClasseAbstrata.
module meuPacote_ClasseConcreta_Class

use meuPacote_ClasseAbstrata_Class

implicit none

! acessibilidade padrão:
private

! esta classe estende ClasseAbstrata:
type, public :: ClasseConcreta

    ! atributos no escopo da instância:
    private
    type(ClasseAbstrata) :: super

end type ClasseConcreta

! métodos públicos:
public :: new, finish, getAtributo, setAtributo
public :: metodo1, metodo2

! interfaces para métodos no escopo da instância:

interface new
    module procedure ClasseConcreta_new
end interface new

interface finish
    module procedure ClasseConcreta_finish
end interface finish

interface getAtributo
    module procedure ClasseConcreta_getAtributo
end interface getAtributo

interface setAtributo
    module procedure ClasseConcreta_setAtributo
end interface setAtributo

interface metodo1
    module procedure ClasseConcreta_metodo1
end interface metodo1

interface metodo2
    module procedure ClasseConcreta_metodo2
end interface metodo2
```

```
contains

!********************
!* métodos públicos *
!********************

! construtor:
subroutine ClasseConcreta_new(this,valor)
type(ClasseConcreta), intent(inout) :: this
real, intent(in) :: valor

call setAtributo(this,valor)

end subroutine ClasseConcreta_new

! destrutor:
subroutine ClasseConcreta_finish(this)
type(ClasseConcreta), intent(inout) :: this

end subroutine ClasseConcreta_finish

function ClasseConcreta_getAtributo(this) result(valor)
type(ClasseConcreta), intent(in) :: this
real :: valor

valor = getAtributo(this%super)

end function ClasseConcreta_getAtributo

subroutine ClasseConcreta_setAtributo(this,valor)
type(ClasseConcreta), intent(inout) :: this
real, intent(in) :: valor

call setAtributo(this%super,valor)

end subroutine ClasseConcreta_setAtributo

subroutine ClasseConcreta_metodo1(this)
type(ClasseConcreta), intent(inout) :: this

call metodo1(this%super)

end subroutine ClasseConcreta_metodo1

function ClasseConcreta_metodo2(this) result(numero)
type(ClasseConcreta), intent(in) :: this
real :: numero

numero = getAtributo(this) * getAtributo(this)

end function ClasseConcreta_metodo2

end module meuPacote_ClasseConcreta_Class
```

C6. meuPacote_Package.f90

```
! Pacote com todas as classes e interfaces.
module meuPacote_Package

! classes públicas continadas no pacote:
use meuPacote_Ancestral_Class
use meuPacote_Descendente_Class
use meuPacote_Funcao_Interface
use meuPacote_ClasseAbstrata_Class
use meuPacote_ClasseConcreta_Class

! acessibilidade padrão:
public

end module meuPacote_Package
```

C7. teste1.f90

```
program teste1

use meuPacote_Package  ! importa todas as classes em meuPacote

implicit none

type(Ancestral)   :: ancestral1,   ancestral2
type(Descendente) :: descendente1, descendente2
type(Funcao) :: f

call new(ancestral1,1.0)
call new(ancestral2,2.0)
call new(descendente1,3.0,130.0)
call new(descendente2,4.0,140.0)

write (6,*) Ancestral_getNInstancias(), getAtributo1(ancestral1)
write (6,*) Ancestral_getNInstancias(), getAtributo1(ancestral2)
write (6,*) Descendente_getNInstancias(), getAtributo1(descendente1), &
            & getAtributo2(descendente1)
write (6,*) Descendente_getNInstancias(), getAtributo1(descendente2), &
            & getAtributo2(descendente2)

call finish(ancestral1)
call finish(ancestral2)
call finish(descendente1)
call finish(descendente2)

call new(f,"Ancestral",1.0)
write (6,*) avalie(f,2.0)
call finish(f)

call new(f,"Descendente",3.0,4.0)
write (6,*) avalie(f,2.0)
call finish(f)

end program teste1
```

Saída:

```
           4   1.000000
           4   2.000000
           4   3.000000       130.0000
           4   4.000000       140.0000
   2.000000
   10.00000
```

C8. teste2.f90

```
program teste2

use meuPacote_ClasseConcreta_Class

implicit none

type(ClasseConcreta) :: minhaClasse

call new(minhaClasse,2.0)
call metodo1(minhaClasse)
call finish(minhaClasse)

end program teste2
```

Saída

```
0.0000000E+00
```

C9. Ancestral.java

```
package meuPacote;

public class Ancestral implements Funcao {

    private static int nInstancias = 0;
    protected float atributo1;

    // Construtor:
    public Ancestral(float valor1) {
        setAtributo1(valor1);
        incNInstancias();
    }

    // "Destrutor":
    public void finish() {
        decNInstancias();
    }

    public static int getNInstancias() {
        return nInstancias;
    }

    public float getAtributo1() {
        return atributo1;
    }

    public void setAtributo1(float valor1) {
        atributo1 = valor1;
    }

    public float avalie(float x) {
        return x * atributo1;
    }

    private static void incNInstancias() {
        nInstancias++;
    }

    private static void decNInstancias() {
        nInstancias--;
    }

}
```

C10. Descendente.java

```
package meuPacote;

public class Descendente extends Ancestral implements Funcao {

    protected float atributo2;

    // Construtor:
    public Descendente (float valor1, float valor2) {
        super(valor1);  // chama o construtor da classe ancestral
        setAtributo2(valor2);
    }

    public float getAtributo2() {
        return atributo2;
    }

    public void setAtributo2(float valor2) {
        atributo2 = valor2;
    }

    public float avalie(float x) {
        return x * (float)Math.sqrt(atributo1*atributo1 + atributo2*atributo2);
    }

}
```

C11. Funcao.java

```
package meuPacote;

public interface Funcao {

    float avalie(float x);

}
```

C12. ClasseAbstrata.java

```
package meuPacote;

public abstract class ClasseAbstrata {

    protected float atributo;

    public float getAtributo() {
        return atributo;
    }

    public void setAtributo(float valor) {
        atributo = valor;
    }

    public void metodo1() {
        System.out.println(metodo2());
    }

    protected abstract float metodo2();

}
```

C13. ClasseConcreta.java

```
package meuPacote;

public class ClasseConcreta extends ClasseAbstrata {

    // Construtor:
    public ClasseConcreta(float valor) {
        atributo = valor;
    }

    protected float metodo2() {
        return atributo*atributo;
    }

}
```

C14. Teste1.java

```
package meuPacote;

public class Teste1 {

    public static void main(String[] args) {

        Ancestral ancestral1 = new Ancestral(1.0F);
        Ancestral ancestral2 = new Ancestral(2.0F);
        Descendente descendente1 = new Descendente(3.0F,130.0F);
        Descendente descendente2 = new Descendente(4.0F,140.0F);

        System.out.println( ancestral1.getNInstancias() + " " +
                            ancestral1.getAtributo1() );
        System.out.println( ancestral2.getNInstancias() + " " +
                            ancestral2.getAtributo1() );
        System.out.println( descendente1.getNInstancias() + " " +
                            descendente1.getAtributo1()   + " " +
                            descendente1.getAtributo2() );
        System.out.println( descendente2.getNInstancias() + " " +
                            descendente2.getAtributo1()   + " " +
                            descendente2.getAtributo2());

        ancestral1.finish();
        ancestral2.finish();
        descendente1.finish();
        descendente2.finish();

        Funcao f = new Ancestral(1.0F);
        System.out.println(f.avalie(2.0F));
        f = new Descendente(3.0F,4.0F);
        System.out.println(f.avalie(2.0F));

    }

}
```

Saída

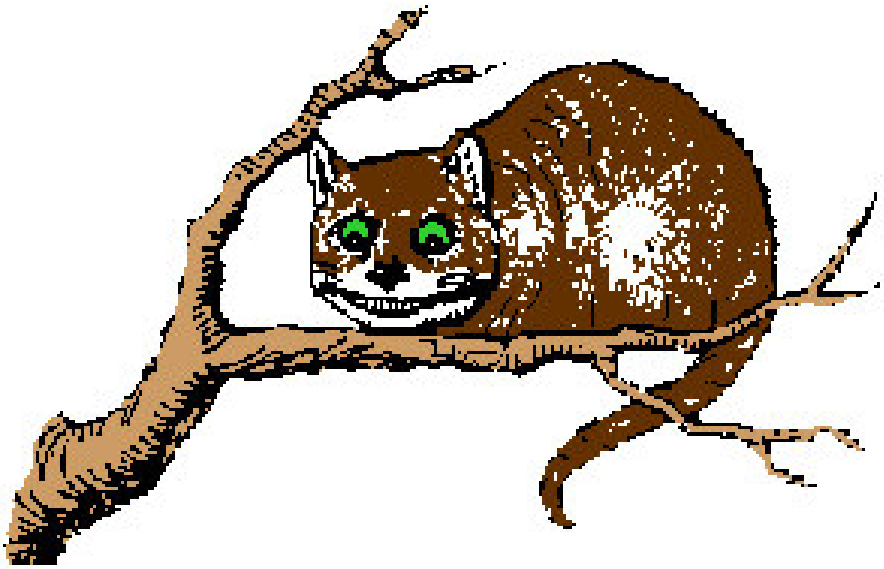
```
4 1.0
4 2.0
4 3.0 130.0
4 4.0 140.0
2.0
10.0
```

C15. Teste2.java

```
package meuPacote;

public class Teste2 {

    public static void main(String[] args) {
        ClasseConcreta minhaClasse = new ClasseConcreta(2.0F);
        minhaClasse.metodo1();
    }

}
```

Saída

```
4.0
```



(...) when she was a little startled by seeing the Cheshire Cat sitting on a bough of a tree a few yards off.

The Cat only grinned when it saw Alice. It looked good-natured, she thought: still it had *very* long claws and a great many teeth, so she felt that it ought to be treated with respect.

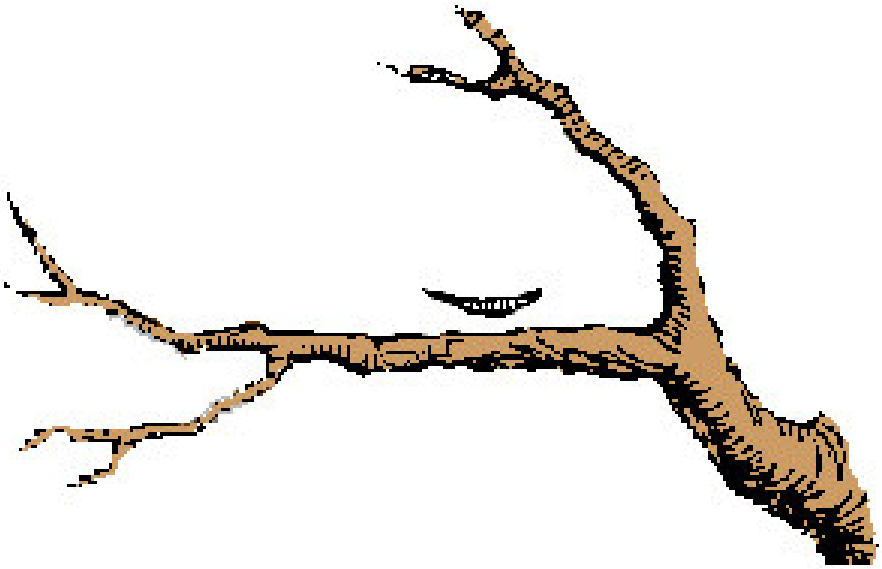
“Cheshire Puss,” she began, rather timidly, as she did not at all know whether it would like the name: however, it only grinned a little wider. “Come, it's pleased so far,” thought Alice, and she went on. “Would you tell me, please, which way I ought to go from here?”

“That depends a good deal on where you want to get to,” said the Cat.

“I don't much care where...” said Alice.

“Then it doesn't matter which way you go,” said the Cat.

“...so long as I get *somewhere*,” Alice added as an explanation.

“Oh, you're sure to do that,” said the Cat, “if you only walk long enough.”

Extraído de “*Alice’s Adventures in Wonderland*”, de Lewis Carroll, edição disponível no Projeto Gutenberg, http://gutenberg.net .