

# Note sul programma “deutone” teoria e pratica

Claudio Bonati

14 novembre 2011

## Parte I Teoria

### 1 Le equazioni da risolvere

Se si suppone che l'interazione tra due nucleoni soddisfi le seguenti proprietà

1. deriva da un potenziale
2. il potenziale è indipendente dalle velocità delle particelle
3. il potenziale è uno scalare (sotto Lorentz e parità)

la forma generale che può assumere il potenziale è la seguente

$$V(\mathbf{r}) = V_1(r) + V_2(r)\vec{\sigma}_1\vec{\sigma}_2 + V_3(r)\left(\frac{3}{r^2}(\vec{\sigma}_1\mathbf{r})(\vec{\sigma}_2\mathbf{r}) - \vec{\sigma}_1\vec{\sigma}_2\right) \quad (1.1)$$

dove le  $V_i$  sono funzioni generiche della distanza  $r = |\mathbf{r}|$  e

$$\begin{aligned}\vec{\sigma}_1 &= (\sigma_x, \sigma_y, \sigma_z) \text{ per la particella 1} \\ \vec{\sigma}_2 &= (\sigma_x, \sigma_y, \sigma_z) \text{ per la particella 2}\end{aligned} \quad (1.2)$$

Il termine che moltiplica  $V_3$  è spesso indicato con  $S_{12}$  ed è tipicamente indicato con il nome di termine tensoriale. Se indichiamo con  $\mathbf{S}$  lo spin totale  $\mathbf{S} = \frac{1}{2}(\vec{\sigma}_1 + \vec{\sigma}_2)$ , con  $\mathbf{L}$  il momento angolare orbitale ( $\mathbf{L} = \mathbf{r} \times \mathbf{p}$ ) e con  $\mathbf{J} = \mathbf{S} + \mathbf{L}$  il momento angolare totale, è semplice vedere che

$$\begin{aligned}S_{12} &= \frac{6}{r^2}(\mathbf{S}\mathbf{r})^2 - 2\mathbf{S}^2 & [S_{12}, \mathbf{S}^2] &= 0 & [S_{12}, S_z] &\neq 0 \\ [S_{12}, \mathbf{L}^2] &\neq 0 & [S_{12}, \mathbf{J}^2] &= 0 & [S_{12}, J_z] &= 0\end{aligned} \quad (1.3)$$

cioè  $S_{12}$  è uno scalare per rotazioni (in quanto commuta con  $\mathbf{J}$ ) e il momento angolare totale e la sua proiezione sull'asse  $z$  sono buoni numeri quantici, lo spin totale è un buon numero quantico ma non la sua proiezione lungo  $z$ , e non lo è il momento angolare orbitale.

Poichè il momento angolare orbitale non è un buon numero quantico, la soluzione del problema sarà in generale una sovrapposizione di stati con vari momenti angolari orbitali.

Nel caso del deutone lo spin vale  $s = 1$  e il momento totale  $j = 1$ , quindi sono permessi i due valori  $\ell = 0$  e  $\ell = 2$ . Se si introducono le funzioni  $\mathcal{Y}_{\ell s j j_z}$  che descrivono la componente  $\ell$  e  $s$  dello stato con momento totale  $j$  e proiezione  $j_z$ , si ha

$$\mathcal{Y}_{\ell s j j_z} = \sum_{m+s_z=j_z} \langle \ell m s s_z | j j_z \rangle Y_{\ell m} \chi_{s s_z} \quad (1.4)$$

dove  $Y_{\ell m}$  sono le armoniche sferiche e  $\chi_{ss_z}$  le funzioni d'onda di spin. La funzione d'onda del deutone può quindi essere scritta nella forma

$$\Psi = \frac{u(r)}{r} \mathcal{Y}_{011j_z} + \frac{w(r)}{r} \mathcal{Y}_{211j_z} \quad (1.5)$$

dove esplicitamente si hanno le espressioni

$$\begin{aligned} \mathcal{Y}_{011j_z} &= Y_{00} \chi_{1j_z} \\ \mathcal{Y}_{211j_z} &= \sqrt{\frac{(3+j_z)(2+j_z)}{20}} Y_{2j_z+1} \chi_{1-1} - \sqrt{\frac{(2-j_z)(2+j_z)}{10}} Y_{2j_z} \chi_{10} + \\ &\quad + \sqrt{\frac{(2-j_z)(3-j_z)}{20}} Y_{2j_z-1} \chi_{11} \end{aligned} \quad (1.6)$$

Per come sono state definite le funzioni  $\mathcal{Y}_{\ell s j j_z}$  sono ortonormali quando integrate sugli angoli, quindi la condizione di normalizzazione per la funzione d'onda del deutone si scrive

$$\int (u(r)^2 + w(r)^2) dr = 1 \quad (1.7)$$

Usando le Eq. (1.6) si possono verificare direttamente le identità

$$\begin{aligned} S_{12} \mathcal{Y}_{011j_z} &= \sqrt{8} \mathcal{Y}_{011j_z} \\ S_{12} \mathcal{Y}_{211j_z} &= \sqrt{8} \mathcal{Y}_{011j_z} - 2 \mathcal{Y}_{211j_z} \end{aligned} \quad (1.8)$$

utilizzando le quali la funzione d'onda del deutone può essere riscritta nella forma coincisa

$$\Psi = \frac{1}{4\pi} \left( \frac{u(r)}{r} + \frac{1}{\sqrt{8}} \frac{w(r)}{r} S_{12} \right) \chi_{1j_z} \quad (1.9)$$

L'equazione di Schrodinger per il deutone, descritto tramite una interazione del tipo (1.1), è un sistema di due equazioni accoppiate:

$$\begin{aligned} -\frac{\hbar^2}{2\mu} \frac{d^2 u}{dr^2} + V_c(r)u(r) + \sqrt{8}V_t(r)w(r) &= Eu(r) \\ -\frac{\hbar^2}{2\mu} \left( \frac{d^2 w}{dr^2} - \frac{6}{r^2} w(r) \right) + [V_c(r) - 2V_t(r)]w(r) + \sqrt{8}V_t(r)u(r) &= Ew(r) \end{aligned} \quad (1.10)$$

dove si sono introdotte le notazioni

$$V_c(r) = V_1(r) + V_2(r) \quad V_t(r) = V_3(r) \quad (1.11)$$

e  $\mu$  è la massa ridotta del sistema neutrone protone.

Una volta ottenute le funzioni  $u$  e  $w$  si possono calcolare alcune osservabili di interesse fisico, quali il raggio medio  $r_d$ , il momento magnetico medio  $\mu_d$  ed il momento di quadrupolo elettrico  $Q_d$ , tramite le espressioni

$$\begin{aligned} r_d &= \frac{1}{2} \int r(u(r)^2 + w(r)^2) dr \\ \mu_d &= \mu_n + \mu_p - \frac{3}{2} \left( \mu_n + \mu_p - \frac{1}{2} \right) \int w(r)^2 dr \\ Q_d &= \frac{\sqrt{2}}{10} \int r^2 u(r) w(r) dr - \frac{1}{20} \int r^2 w(r)^2 dr \end{aligned} \quad (1.12)$$

## 2 Il metodo di risoluzione

Se indichiamo con  $\psi$  il vettore

$$\psi = \begin{pmatrix} u \\ w \end{pmatrix} \quad (2.1)$$

allora le equazioni Eq. (1.10) possono essere scritte nella forma  $H\psi = E\psi$  con

$$H = \begin{pmatrix} -\frac{\hbar}{2\mu} \frac{d^2}{dr^2} + V_c(r) & \sqrt{8}V_t(r) \\ \sqrt{8}V_t(r) & -\frac{\hbar^2}{2\mu} \left( \frac{d^2}{dr^2} - \frac{6}{r^2} \right) + V_c(r) - 2V_t(r) \end{pmatrix} \quad (2.2)$$

e l'operatore  $H$  è autoaggiunto. Questo sistema agli autovalori può essere risolto in molti modi, noi procederemo con quello che è probabilmente il più rozzo ma il più diretto e semplice da implementare: il metodo delle differenze finite. In estrema sintesi questo significa discretizzare il problema utilizzando una griglia ed approssimare le derivate che compaiono nella definizione di  $H$  con delle differenze finite.

Consideriamo una griglia di passo uniforme  $\Delta$ , allora la derivata seconda può essere scritta come

$$\frac{d^2 f}{dx^2} = \frac{f(x + \Delta) + f(x - \Delta) - 2f(x)}{\Delta^2} + O(\Delta^2) \quad (2.3)$$

quindi, trascurando termini di ordine superiore in  $\Delta$  la matrice di  $\frac{d^2}{dx^2}$  può essere scritta come

$$\frac{d^2}{dx^2} = \frac{1}{\Delta^2} \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ \cdots & 1 & -2 & 1 & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 1 & -2 & 1 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & -2 & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 1 & -2 & 1 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & 1 & -2 & 1 & \cdots \\ \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad (2.4)$$

e questa matrice è infinita. Un poco di cura deve essere prestata nel troncarla, ovvero si devono usare le condizioni al bordo delle equazioni Eq. (1.10). Per  $x = 0$  sia  $u$  che  $w$  devono annullarsi, quindi una possibilità è usare come griglia di discretizzazione l'insieme dei punti  $x_n = n\Delta$  con  $n > 0$ , in quanto in questo modo, usando le condizioni al bordo (b.c.), la derivata seconda in  $x_1$  si scriverebbe

$$f''(x_1) = \frac{f(x_2) + f(x_1 - \Delta) - 2f(x_1)}{\Delta^2} \stackrel{\text{b.c.}}{=} \frac{f(x_2) - 2f(x_1)}{\Delta^2} \quad (2.5)$$

quindi la derivata seconda discretizzata con le condizioni al bordo  $f(0) = 0$  sulla griglia  $x_n = n\Delta$ ,  $n > 0$ , diventa

$$\left. \frac{d^2}{dx^2} \right|_{f(0)=0} = \frac{1}{\Delta^2} \begin{pmatrix} -2 & 1 & 0 & 0 & 0 & 0 & \cdots \\ 1 & -2 & 1 & 0 & 0 & 0 & \cdots \\ 0 & 1 & -2 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 1 & -2 & 1 & 0 & \cdots \\ 0 & 0 & 0 & 1 & -2 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad (2.6)$$

che è ora solo semi-infinita.

La condizione al bordo all'infinito non è semplicemente implementabile nella discretizzazione alle differenze finite del problema agli autovalori Eq. (1.10) e la cosa più semplice è scegliere un  $R$  sufficientemente grande in modo che  $u''(R) \sim u(R) \sim 0$  e  $w''(R) \sim w(R) \sim 0$  ed utilizzare come

seconda condizione al limite  $f(R) = 0$ , cioè

$$\left. \frac{d^2}{dx^2} \right|_{\substack{f(0)=0 \\ f(R)=0}} = \frac{1}{\Delta^2} \begin{pmatrix} -2 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & -2 \end{pmatrix} \quad (2.7)$$

i cui elementi di matrice sono

$$\left[ \left. \frac{d^2}{dx^2} \right|_{\substack{f(0)=0 \\ f(R)=0}} \right]_{ij} = \frac{1}{\Delta^2} (-2\delta_{ij} + \delta_{1|i-j|}) \quad (2.8)$$

Chiaramente alla fine del calcolo si dovrà verificare la stabilità del risultato al variare di  $R$ , ovvero di aver scelto  $R$  sufficientemente grande.

A questo punto le equazioni Eq. (1.10) sono state discretizzate su una griglia finita e la corrispondente matrice  $H$  si vede subito essere reale simmetrica, quindi può essere diagonalizzata con metodi standard. Il più semplice di questi metodi è il metodo di Jacobi<sup>1</sup>: esporremo qui le idee generali del metodo rimandando ad un qualunque testo di analisi numerica per la dimostrazione che il metodo funziona.

Indichiamo con  $G$  una matrice di rotazione di Givens, ovvero una rotazione di due variabili coordinate scelte tra le  $n$  disponibili. In particolare la matrice di Givens  $G^{\alpha\beta}$  ( $\alpha < \beta$ ) ha componenti

$$\begin{aligned} [G^{\alpha\beta}]_{ij} &= \delta_{ij} \quad \text{se } i \neq \alpha, \beta \\ [G^{\alpha\beta}]_{\alpha j} &= \delta_{\alpha j} \cos \theta - \delta_{\beta j} \sin \theta \\ [G^{\alpha\beta}]_{\beta j} &= \delta_{\alpha j} \sin \theta + \delta_{\beta j} \cos \theta \end{aligned} \quad (2.9)$$

Data la matrice simmetrica  $A$  consideriamo allora  $\tilde{A} = (G^{\alpha\beta})^T A G^{\alpha\beta}$  che ha elementi

$$\begin{aligned} \tilde{A}_{\alpha\alpha} &= \cos^2 \theta A_{\alpha\alpha} - 2 \sin \theta \cos \theta A_{\alpha\beta} + \sin^2 \theta A_{\beta\beta} \\ \tilde{A}_{\beta\beta} &= \sin^2 \theta A_{\alpha\alpha} + 2 \sin \theta \cos \theta A_{\alpha\beta} + \cos^2 \theta A_{\beta\beta} \\ \tilde{A}_{\alpha\beta} &= \tilde{A}_{\beta\alpha} = (\cos^2 \theta - \sin^2 \theta) A_{\alpha\beta} + \sin \theta \cos \theta (A_{\alpha\alpha} - A_{\beta\beta}) \\ \tilde{A}_{\alpha k} &= \tilde{A}_{k\alpha} = \cos \theta A_{\alpha k} - \sin \theta A_{\beta k} \quad k \neq \alpha, \beta \\ \tilde{A}_{\beta k} &= \tilde{A}_{k\beta} = \sin \theta A_{\alpha k} + \cos \theta A_{\beta k} \quad k \neq \alpha, \beta \\ \tilde{A}_{jk} &= \tilde{A}_{kj} = A_{jk} \quad j, k \neq \alpha, \beta \end{aligned} \quad (2.10)$$

Se si sceglie

$$\tan(2\theta) = \frac{2A_{\alpha\beta}}{A_{\alpha\alpha} - A_{\beta\beta}} \quad (2.11)$$

allora dalle Eq. (2.10) segue che  $\tilde{A}_{\alpha\beta} = \tilde{A}_{\beta\alpha} = 0$ . Il metodo di Jacobi consiste nell'utilizzare iterativamente questo metodo per eliminare tutti gli elementi fuori diagonale di  $A$ . Se si tiene conto delle matrici di Givens usate nel procedimento, quando la matrice  $A$  è stata diagonalizzata dal prodotto di tutte le matrici di Givens si possono ottenere gli autovettori di  $A$ .

---

<sup>1</sup>Questo metodo risulta efficiente fintanto che la dimensione della matrice è  $O(100)$ . Per matrici più grandi i metodi tipicamente usati sono quelli che usano le decomposizioni  $QR$ , che sono però molto meno immediati da programmare e capire

## Parte II

# Pratica

### 3 Descrizione del programma

I files che compongono il programma sono contenuti nella cartella **src** e sono

**deutone.c** il **main** del programma, che alloca e libera le risorse e chiama le funzioni per i vari calcoli.

**diagonalize.c** la routine di diagonalizzazione con il metodo di Jacobi

**global\_var.h** l'header file in cui sono contenute le definizioni/dichiarazioni delle variabili globali (ciò comuni a tutti i files)

**matrix\_init.c** la funzione che inizializza la matrice da diagonalizzare secondo lo schema dato da Eq. (2.2)-(2.9)

**observables.c** le funzioni per il calcolo delle osservabili Eq. (1.12)

**potential.c** le funzioni che definiscono i potenziali (scalare, di spin, tensore)

**read\_input.c** la funzione che legge da input i valori

oltre ad altri header files che contengono le dichiarazioni delle funzioni.

### 4 Compilazione

Sui sistemi Unix (Linux e Mac) la compilazione può essere eseguita come segue, su macchine Windows è prima necessario installare un emulatore Linux, come ad esempio cygwin. Aprendo una shell nella cartella iniziale (quella in cui è presente il file **configure**) si deve digitare

```
./configure
```

e premere invio. Si dovrebbe ottenere un output a schermo del tipo di quello di Fig. (1). Dopo di ciò si deve digitare

```
make
```

e premere invio. Un output simile a quello in Fig. (2) dovrebbe essere ottenuto. Se tutto è funzionato bene nella cartella **src** dovrebbe essere uscito un file eseguibile chiamato **deutone** (su Windows si dovrebbe chiamare **deutone.exe**, di conseguenza su Windows in tutti i comandi successivi **./deutone** deve essere sostituito da **./deutone.exe**). Se a questo punto si entra nella cartella **src** e si digita

```
./deutone
```

premendo poi invio si dovrebbe ottenere l'output mostrato in Fig. (3) ed il file **input\_template** dovrebbe essere stato scritto. Questo file è il file di ingresso per il programma e può essere modificato senza ricompilare. I valori iniziali sono regolati in modo da ottenere valori abbastanza realistici per le osservabili. Per eseguire il programma si deve a questo punto eseguire

```
./deutone input_template
```

Sullo schermo vengono stampati tra l'altro i livelli energetici ottenuti (nel caso del template solo uno) insieme alle osservabili calcolate per lo stato fondamentale. Vengono inoltre stampati due files:

`fundamental_psi.dat` la colonna 1 rappresenta il raggio  $r$  in fermi, le altre due colonne sono rispettivamente le componenti a momento angolare angolare 0 e 2 della funzione d'onda del fondamentale.

`potential.dat` la dipendenza radiale del potenziale centrale  $V_c$  e del termine tensore  $V_t$

Le funzioni che definiscono il potenziale possono essere modificate semplicemente modificando il file `potential.c` e poi rieseguendo “make” per ricompilare.

#### 4.1 Nota per i più pratici di programmazione

Nel caso in cui si volessero far effettuare al compilatore particolari ottimizzazioni dipendenti dalla macchina si può usare la variabile `CFLAGS` nella configurazione: ad esempio sul mio computer i tempi migliori sono ottenuti usando

```
./configure CFLAGS="-O3 -m64 -mmmx -msse -msse2 -mssse3 -msse4 -ffast-math"
```

questo richiede però che la CPU usata sia a 64 bit e che abbia i registri SIMD `mmx`, `sse`, `sse2`, `ssse3` e `sse4`. Per verificare la presenza di questi (o altri) dettagli è comodo consultare il file `\proc\cpuinfo`.

### 5 Esempio di uso

A parte i parametri del potenziale, nel file di input sono specificate due costanti di importanza fondamentale per il funzionamento del programma: la prima rappresenta il numero di punti di discretizzazione ed è indicata nel programma con `num`, la seconda è il punto  $R$  della sezione 2 ed è indicata nel programma con `dist_infinity`. Chiaramente i risultati del programma non sono esatti ma hanno diverse fonti di errore:

- errori puramente numerici. Tutto il programma è scritto in doppia precisione, quindi questo tipo di errore influisce tipicamente sulle quindicesime cifre significative dei risultati e sarà ignorato nel seguito.
- errori di discretizzazione, dovuti al fatto che `num`  $< \infty$ .
- errori di condizione al bordo, dovuti al fatto che `dist_infinity`  $< \infty$ .

Per chi fosse curioso di vedere esplicitamente gli errori numerici della diagonalizzazione, utilizzando al posto del semplice “configure”

```
./configure --enable-debug-mode
```

alla fine della diagonalizzazione si verifica esplicitamente la precisione numerica degli autovalori e autovettori, calcolando  $\|Hv_i - \lambda_i v_i\|$ .

Consideriamo prima di tutto gli errori di discretizzazione: come si vede da Eq. (2.3) nella discretizzazione abbiamo commesso un errore  $O(\Delta^2)$ , quindi ci aspettiamo che, a `dist_infinity` fissato, i valori ottenuti  $O$  abbiano una dipendenza del tipo

$$O = A + B\text{num}^{-2} + o(\text{num}^{-2}) \quad (5.1)$$

che è esplicitamente verificata in Fig. (4). Utilizzando questa espressione si può fittare il valore asintotico di una osservabile nel limite continuo  $\Delta \rightarrow 0$ . Questo valore dipenderà ancora da `dist_infinity`, tuttavia in questo caso la dipendenza è molto più benigna: poichè le funzioni d'onda degli stati legati decadono esponenzialmente a grande distanza, ci aspettiamo che la dipendenza residua da `dist_infinity` sia esponenziale, fatto verificato direttamente in Fig. (5).

Ci si può a questo punto chiedere quanto costoso da un punto di vista computazionale sia aumentare le precisioni. Per analizzare questo fatto è prima di tutto necessario analizzare la

```

claudio@claudio-pc:~/Documents/programmic/deutone$ ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
checking for sqrt in -lm... yes
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking math.h usability... yes
checking math.h presence... yes
checking for math.h... yes
checking stdio.h usability... yes
checking stdio.h presence... yes
checking for stdio.h... yes
checking for stdlib.h... (cached) yes
checking for stdlib.h... (cached) yes
checking for GNU libc compatible malloc... yes
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands

```

Figura 1: Risultato del “configure” sul mio computer.

```

claudio@claudio-pc:~/Documents/programmic/deutone$ make
make all-recursive
make[1]: Entering directory '/home/claudio/Documents/programmic/deutone'
Making all in src
make[2]: Entering directory '/home/claudio/Documents/programmic/deutone/src'
gcc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT deutone.o -MD -MP -MF
.deps/deutone.Tpo -c -o deutone.o deutone.c
mv -f .deps/deutone.Tpo .deps/deutone.Po
gcc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT diagonalize.o -MD -MP -MF
.deps/diagonalize.Tpo -c -o diagonalize.o diagonalize.c
mv -f .deps/diagonalize.Tpo .deps/diagonalize.Po
gcc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT matrix_init.o -MD -MP -MF
.deps/matrix_init.Tpo -c -o matrix_init.o matrix_init.c
mv -f .deps/matrix_init.Tpo .deps/matrix_init.Po
gcc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT observables.o -MD -MP -MF
.deps/observables.Tpo -c -o observables.o observables.c
mv -f .deps/observables.Tpo .deps/observables.Po
gcc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT potential.o -MD -MP -MF
.deps/potential.Tpo -c -o potential.o potential.c
mv -f .deps/potential.Tpo .deps/potential.Po
gcc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT read_input.o -MD -MP -MF
.deps/read_input.Tpo -c -o read_input.o read_input.c
mv -f .deps/read_input.Tpo .deps/read_input.Po
gcc -g -O2      -o deutone deutone.o diagonalize.o matrix_init.o
observables.o potential.o read_input.o -lm
make[2]: Leaving directory '/home/claudio/Documents/programmic/deutone/src'
make[2]: Entering directory '/home/claudio/Documents/programmic/deutone'
make[2]: Leaving directory '/home/claudio/Documents/programmic/deutone'
make[1]: Leaving directory '/home/claudio/Documents/programmic/deutone'

```

Figura 2: Risultato del “make” sul mio computer.

```

Program deutone version 1.3
Claudio Bonati, bonati@df.unipi.it
Use: ./deutone input_file
Example of input_file: input_template

```

Figura 3: Risultato del comando “./deutone” sul mio computer.



Dipendenza di E da num a dist\_infinity fissata

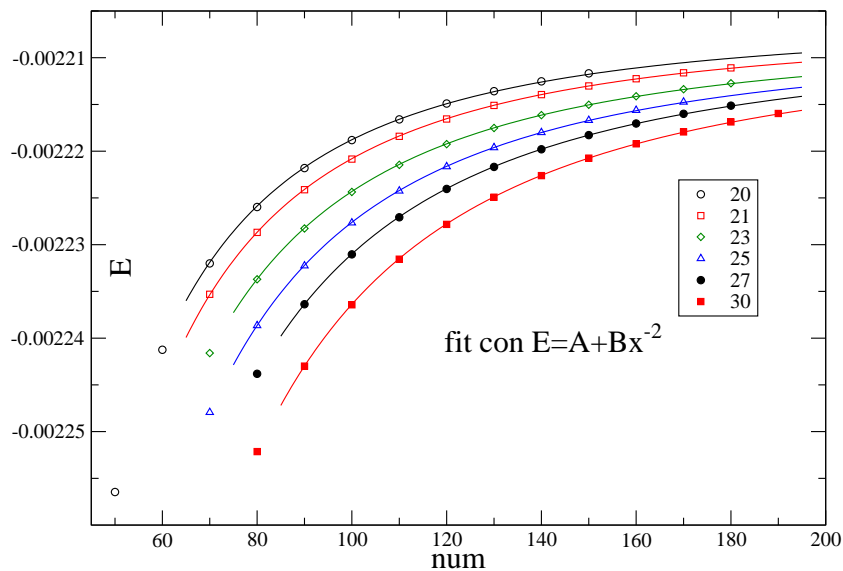


Figura 4: Errori di discretizzazione.

Valori di E estrapolati per  $\text{num} \rightarrow \text{infinito}$

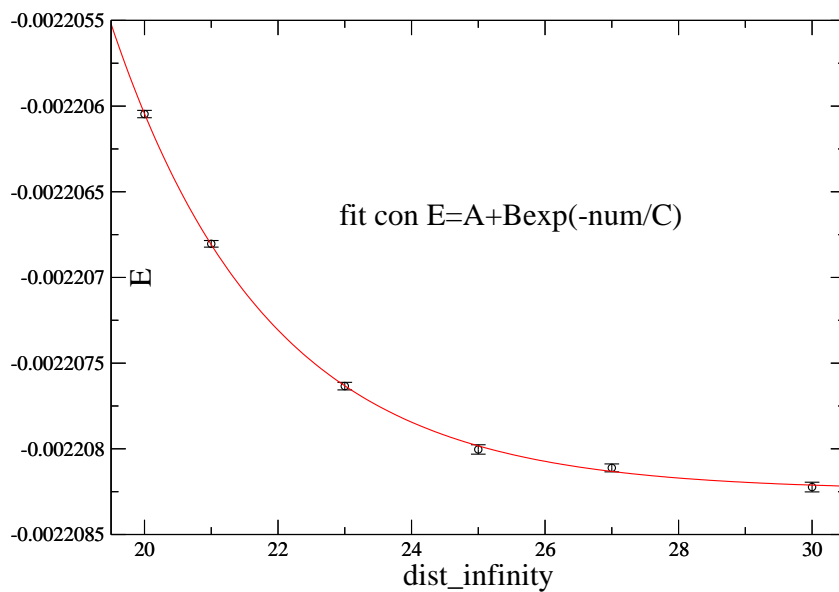


Figura 5: Errori sulla condizione al bordo all'infinito.

dipendenza del tempo di soluzione dai dati del problema, tipicamente dal valore di `num`. È un fatto generale che tutti gli algoritmi di diagonalizzazione esatta per matrici  $N \times N$  hanno un costo computazionale  $O(N^3)$ , inoltre in alcuni casi può essere rilevante anche il tempo necessario per la allocazione fisica della matrice nella memoria del computer, che scala come  $O(N^2)$ , quindi useremo un fit del tipo

$$T = A + B\text{num}^2 + C\text{num}^3 \quad (5.2)$$

che risulta in effetti essere ragionevolmente buono (vedi Fig. (6)). Combinando i risultati di Fig. (4) e Fig. (6) si può a questo punto ottenere una figura in cui il tempo di esecuzione sia espresso in funzione della precisione della soluzione: vedi Fig. (7).

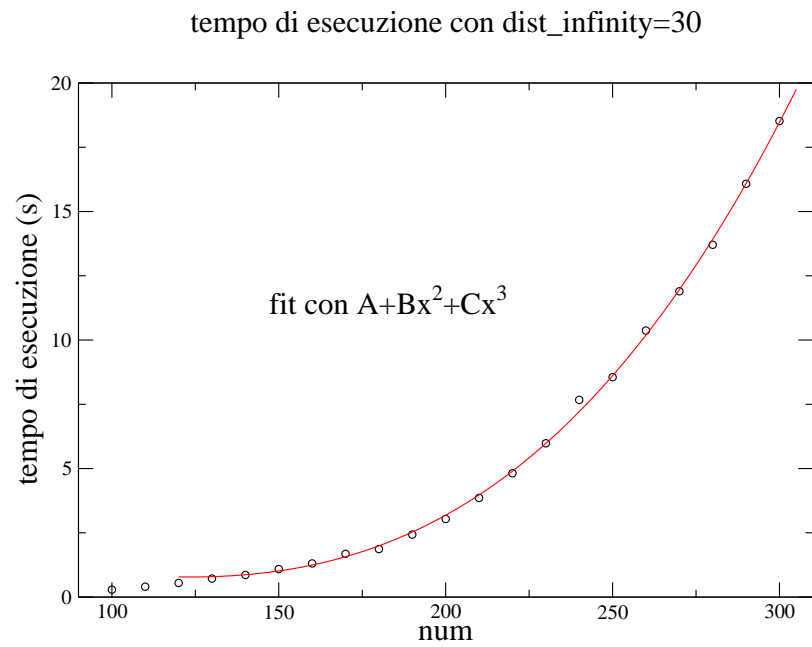


Figura 6: Dipendenza del tempo di esecuzione dalla discretizzazione.

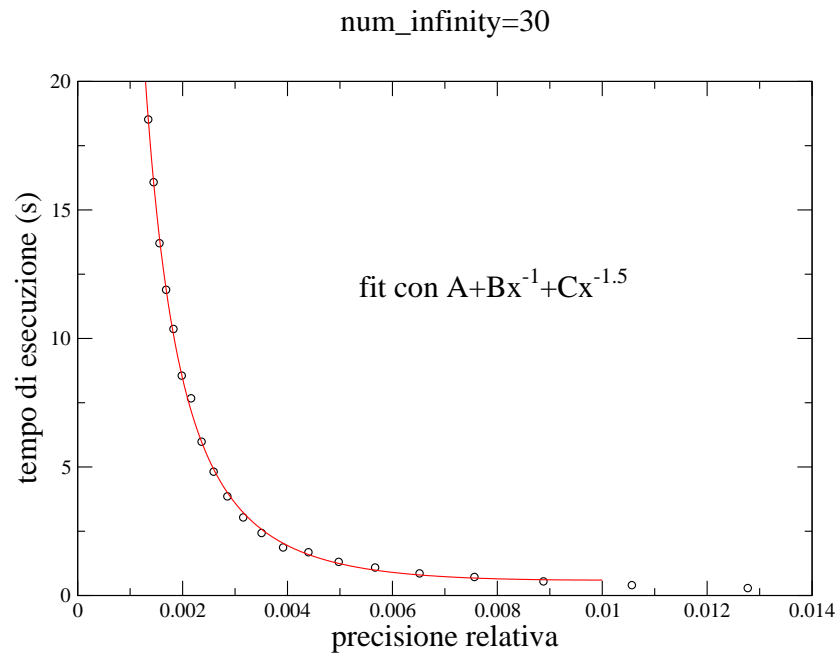


Figura 7: Dipendenza del tempo di esecuzione dalla precisione che si vuole ottenere.