Due date final report: TBD

Computational Physics — Fall 2017

Solving the Nuclear Two-Body Problem with Integral Equations

v1.4e — 20Sep2017 — H. Haberzettl [Wmatrix2017.tex (30Apr2009) — LaTeXed on 20Sep2017, at 09:02]

Preliminary version FYI — Final version will be distributed later

1 Project

Calculate the ***s*-wave phase shifts**, **scattering lengths**, and **bound states (energies and momentum-space wave functions)** for two semi-realistic central nucleon-nucleon (NN) potentials, namely the Malfliet–Tjon I and III potentials given in sec. 3. The potentials MT-I and MT-III, respectively, describe the ${}^1S_0$ and ${}^3S_1$ states of the two-nucleon system, where the notation ${}^{(2S+1)}L_J$ indicates the total spin S, the angular momentum L, and the total angular momentum J of the two-fermion systems.

This is a two-body problem that is reduced to an effective one-body problem, with a reduced mass

$$\mu = \frac{m}{2} \,, \tag{1}$$

where m is the nucleon mass. (I assume here that both neutron and proton have the same mass m.) The problem is to be solved in natural units (where $\hbar = c = 1$) and

$$\frac{\hbar^2}{2\mu} = \frac{\hbar^2}{m} = 41.47\,\text{MeV fm}^2 \,. \tag{2}$$

All of the following relations are to be understood in these units (see also sec. 4). Note, in particular, that — in view of $\hbar = 1$ — there is no difference between momenta and wave numbers.

The method to be used is based on the nonsingular formulation of the scattering problem. The full formalism of how nonsingular integral equations provide solutions for both scattering and bound-state problems is given in my Lecture Notes [1] in terms of quantities called W-matrices [2].

1.1 What to expect

You will find that the so-called *singlet* ($={}^1S_0$) system has no bound state and a relatively large negative scattering length, whereas the *triplet* ($={}^3S_1$) system has one bound state, the deuteron, with a smaller positive scattering length. You will also find that, as a function of the scattering energy, the singlet phase shifts rise sharply from zero, but never reach 180°, and then fall off again back to zero. The triplet phase shifts, by contrast, will start at 180° and then smoothly fall off to zero. These phase-shift findings are in keeping with Levinson's theorem [1].

2 *W*-Matrix Formalism

Here, we will apply the formalism to an s-wave potential $V(r)$. We define

$$U(p,q) = \frac{2}{\pi pq} \int_0^\infty dr\, \sin(pr)[mV(r)]\sin(qr)\ , \tag{3}$$

which is the s-wave Fourier transform of the potential. Some common limits are

$$U(p,0) = U(0,p) = \frac{2}{\pi p} \int_0^\infty dr\, r\,[mV(r)]\sin(pr) \qquad \text{and} \qquad U(0,0) = \frac{2}{\pi} \int_0^\infty dr\, r^2\,[mV(r)]\ . \tag{4}$$

See sec. 4 for how to evaluate $[mV(r)]$ in the present units.

The relation

$$\frac{\sin(i\alpha r)}{i\alpha} = \frac{\sinh(\alpha r)}{\alpha} \tag{5}$$

might be useful for analytic continuations to pure imaginary momenta $k = i\alpha$, with $\alpha \geq 0$.

2.1 Scattering problem

For the scattering energy $E = k^2/m$, the nonsingular W-matrix equation is given by

$$W(p,k) = U(p,k) + \int_0^\infty dq\, q^2\, \frac{U(p,q) - U(p,k)}{k^2 - q^2} W(q,k)\ . \tag{6}$$

Solving this equation, we define the Jost function

$$F(k) = 1 - \int_0^\infty dq\, q^2 \frac{W(q,k)}{k^2 + i0 - q^2}\ , \tag{7}$$

whose real and imaginary parts are

$$\mathrm{Re}\, F(k) = 1 - \int_0^\infty dq\, \frac{q^2 W(q,k) - k^2 W(k,k)}{k^2 - q^2} \qquad \text{and} \qquad \mathrm{Im}\, F(k) = \frac{\pi}{2} k W(k,k)\ , \tag{8}$$

respectively.

Phase shifts $\delta(k)$ and scattering length a then are given by

$$\tan\delta(k) = -\frac{\mathrm{Im}\, F(k)}{\mathrm{Re}\, F(k)} \tag{9}$$

and

$$a = \frac{\pi}{2} \frac{W(0,0)}{1 + \int_0^\infty dq\, W(q,0)}\ , \tag{10}$$

respectively.

2.2 Bound-state problem

The bound-state problem can be solved by eq. (6) with imaginary momentum $k = i\alpha$. However, one can show [2] that one can simplify the problem by considering

$$W(p,\beta) = U(p,\beta) + \int_0^\infty dq\, q^2\, \frac{U(p,q) - U(p,\beta)}{-\alpha^2 - q^2} W(q,\beta) \tag{11}$$

instead, where it is *not* necessary that $\beta = i\alpha$. Here β can be any real parameter. (For example, one may choose $\beta = 0$ for convenience.) Defining then the Jost-like function

$$F_\beta(-\alpha^2) = 1 - \int_0^\infty dq\, q^2 \frac{W(q,\beta)}{-\alpha^2 - q^2}\ , \tag{12}$$

the bound states α_n are found by looking for the roots of $F_\beta(-\alpha^2)$, i.e.,

$$F_\beta(-\alpha_n^2) = 0\ . \tag{13}$$

The bound-state energy is then $E_n = -\alpha_n^2/m$. The analytic continuation of the Jost function (7) for $k = i\alpha$ is given by

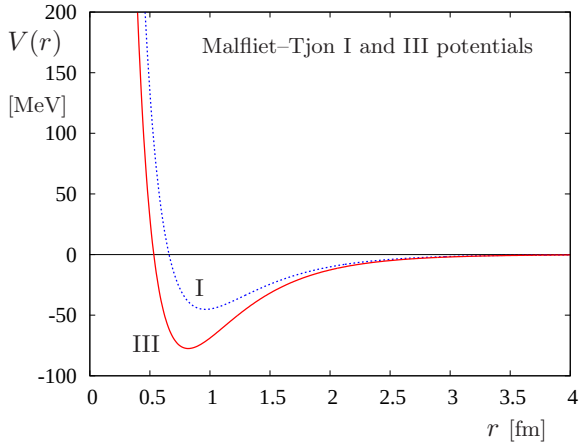
$$F(i\alpha) = F_{i\alpha}(-\alpha^2)\ . \tag{14}$$

3 Malfliet–Tjon I-III potentials

The Malfliet–Tjon potentials [3] for the nucleon-nucleon problem are semi-realistic interactions that combine an attractive potential with a repulsive core in the form of two Yukawa terms, i.e.,

$$V(r) = V_R \frac{\mathrm{e}^{-\mu_R r}}{r} - V_A \frac{\mathrm{e}^{-\mu_A r}}{r}\ . \tag{15}$$

The particular parameter sets used here are shown in the adjacent table together with the resulting graphs. The two parameters sets are referred to as I and III; they respectively model the s-wave configurations of the $^1S_0$ (I) and the $^3S_1$ (III) channels of the NN problem. The potentials differ only in the strength of the attractive part.



MT model	V_R (MeV fm)	μ_R (fm^{-1})	V_A (MeV fm)	μ_A (fm^{-1})
I	1438.720	3.110	513.968	1.550
III	1438.720	3.110	626.885	1.550

Note: *All* digits in these parameters are significant.

4 A Word About Units

The potential V is given in units of MeV since the coefficients V_R and V_A, in MeV fm, are divided by a length (r) given in fm. The units of the product $[mV]$ appearing in eq. (3) thus are

$$[mV] = \frac{1}{41.47\,\mathrm{MeV\,fm}^2} \times \mathrm{MeV} = \frac{1\,\mathrm{fm}^{-2}}{41.47}\ . \tag{16}$$

In other words, the values of the product $[mV]$ are measured in fm^{-2} if the coefficients V_R and V_A as given in the table are divided by 41.47, i.e.,

$$mV_i = \frac{V_i}{41.47\,\mathrm{MeV\,fm}^2}\ , \qquad \text{for} \qquad i = R, A\ . \tag{17}$$

The units of $U(p,q)$ of eq. (3), and thus also those of $W(p,q)$, are then fm.

Taking the nucleon mass as the mass unit, *effectively* this is the same as putting $m = 1$ everywhere and using the conversion

$$41.47\,\mathrm{MeV} = 1\,\mathrm{fm}^{-2} \tag{18}$$

between energy and squared-momentum units. (I say here "effectively" because this does *not* mean that the mass is dimensionless; only that masses are measured in units of the nucleon mass.) This conversion applies to the squared scattering momentum k^2 in eq. (6) as well as to the squared bound-state 'momentum' $-\alpha_n^2$ of eq. (13). Therefore, the *numerical values* of these squared momenta need to be multiplied by 41.47 to obtain the corresponding numerical values in energy units of MeV.

5 Numerical Procedure

5.1 Gaussian integration

Gaussian integration is defined for functions over the interval $[-1,+1]$. To utilize it, all integrations from 0 to ∞ need to be transformed to integrations from -1 to $+1$. For the latter, we then employ a numerical quadrature that turns the integration into a finite summation, i.e.,

$$\int_0^\infty dq\, f(q) = \int_{-1}^{+1} dx\, \frac{dq}{dx} \tilde{f}(x) = \sum_{i=1}^{N} w_i \left[\frac{dq}{dx}\tilde{f}(x)\right]_{x=x_i} + R_N[f,q]\ , \tag{19}$$

with $\tilde{f}(x) = f\big(q(x)\big)$, where $q = q(x)$ is any reasonably smooth *strictly monotonic* transformation function with

$$q(x=-1) = 0 \qquad \text{and} \qquad q(x=+1) = \infty\ . \tag{20}$$

The remainder $R_N[f,q]$ is a functional of the original integrand $f(x)$ and the transformation function $q(x)$. It also depends on the choice of *integration mesh points* and *weights* $\{x_i, w_i\}_{i=1,\ldots,N}$. For so-called *Gaussian integration*, the N pairs (x_i, w_i) are tabulated sets of mesh points and weights (see, e.g., [4]). Popular transformation functions are

$$q(x) = q_0 \frac{1+x}{1-x} \qquad \text{or} \qquad q(x) = q_0 \tan\frac{(1+x)\pi}{4}\ , \tag{21}$$

where $q_0 = q(x = 0)$ is a parameter that divides the mesh in half, i.e., for an even number of mesh points, half the integration points are below q_0 and half above. The integration (19) may thus be written as

$$\int_0^\infty dq\, f(q) = \sum_{i=1}^{N} \tilde{w}_i f_i + R_N[f, q] \;, \tag{22}$$

where

$$f_i = f(q_i) = f\big(q(x_i)\big) \qquad \text{and} \qquad \tilde{w}_i = w_i \left[\frac{dq}{dx}\right]_{x=x_i} . \tag{23}$$

In actual numerical calculations, one drops the remainder $R_N[f, q]$ and chooses an integration mesh that provides stable results within acceptable numerical errors. Typically, one seeks numerical values for the physically relevant parameters that are stable to within 1% (or less) relative error. This is done by trial and error by varying the size N of the integration mesh and the mesh parameter q_0. (Note: Typical mesh sizes for good convergence are somewhere in the range $16 \leq N \leq 32$. Also, for most problems in nuclear physics, it is *very* difficult to reduce the relative error to less than 0.1%.)

5.2 Solving an integral equation

Equations (6) and (11) are inhomogeneous integral equations of the generic structure

$$f(p) = g(p) + \int_0^\infty dq\, K(p, q)\, f(q) \;, \tag{24}$$

where $g(p)$ and $K(p, q)$ are the given inhomogeneity and kernel, respectively, and $f(p)$ is the unknown function to be determined by numerical solution. Using a discrete integration mesh and associated weights and the notation

$$f_i = f(q_i) \;, \qquad g_i = g(q_i) \;, \qquad \text{and} \qquad K_{ij} = K(q_i, q_j)\tilde{w}_j \;, \qquad \text{with} \qquad q_k = q(x_k) \;, \tag{25}$$

this is turned into

$$f_i = g_i + \sum_{j=1}^{N} K_{ij} f_j \qquad \text{or} \qquad \mathbf{f} = \mathbf{g} + \mathsf{K}\mathbf{f} \;, \tag{26}$$

with

$$\mathbf{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} \;, \qquad \mathbf{g} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix} \;, \qquad \text{and} \qquad \mathsf{K} = \begin{pmatrix} K_{11} & K_{12} & \dots & K_{1N} \\ K_{21} & K_{22} & \dots & K_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ K_{N1} & K_{N2} & \dots & K_{NN} \end{pmatrix} . \tag{27}$$

The remainder term of the integration was dropped here since one assumes that the finite integration mesh is sufficiently fine to provide a good approximation of the integral. *(This needs to be tested!)* In matrix notation, this system of linear equations thus reads

$$(\mathbb{1} - \mathsf{K})\, \mathbf{f} = \mathbf{g} \tag{28}$$

which can be solved by inversion, i.e.,

$$\mathbf{f} = (\mathbb{1} - \mathsf{K})^{-1}\, \mathbf{g} \;, \tag{29}$$

where $\mathbb{1}$ is the unit matrix, i.e., $\mathbb{1}_{ij} = \delta_{ij}$. [It is assumed here, of course, that the inverse exists, i.e., that $\det(\mathbb{1} - \mathsf{K}) \neq 0$.]

5.2.1 Iterative solution

In general, integral equations *cannot* be solved iteratively because the resulting series do not converge. However, iteration will work for the W-matrix equations (6) and (11) because of the particular structure of their integral kernels [2]. The corresponding series expansion for the generic equation (26) reads

$$\begin{aligned} \mathbf{f} &= \mathbf{g} + \mathsf{K}\mathbf{g} + \mathsf{K}^2\mathbf{g} + \mathsf{K}^3\mathbf{g} + \cdots \\ &= \mathbf{g} + \mathsf{K}\left(\mathbf{g} + \mathsf{K}\mathbf{g} + \mathsf{K}^2\mathbf{g} + \cdots\right) \end{aligned} \tag{30}$$

which provides the recursive relation

$$\mathbf{f}_n = \mathbf{g} + \mathsf{K}\mathbf{f}_{n-1} \quad \text{for} \quad n \geq 1 \ , \quad \text{with} \quad \mathbf{f}_0 = \mathbf{g} \ . \tag{31}$$

Good convergence can be found for less than 10 iteration steps.

5.3 Using Gauss–Jordan elimination with pivoting

Instead of an explicit inversion as in (29), we shall pursue here the Gauss–Jordan elimination scheme which amounts to an implicit inversion. To this end, note that, generically, eq. (28) can be written as an *inhomogeneous* system of N linear equations,

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N &= b_1 \ , \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2N}x_N &= b_2 \ , \\ a_{31}x_1 + a_{32}x_2 + \cdots + a_{3N}x_N &= b_3 \ , \\ \cdots \qquad\qquad\quad &= \cdots \ , \\ a_{N1}x_1 + a_{N2}x_2 + \cdots + a_{NN}x_N &= b_N \ , \end{aligned} \tag{32}$$

which may be cast in matrix form

$$\mathsf{A}\mathbf{x} = \mathbf{b} \tag{33}$$

for an unknown vector $\mathbf{x}$ with given inhomogeneity vector $\mathbf{b}$ and $N\times N$ coefficient matrix A, i.e.,

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix} \ . \tag{34}$$

The basic idea of the Gauss–Jordan algorithm is to turn this into a system of equations where the coefficient matrix is triangular so that

$$\begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & \dots & a'_{1N} \\ 0 & a'_{22} & a'_{23} & \dots & a'_{2N} \\ 0 & 0 & a'_{33} & \dots & a'_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a'_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_N \end{pmatrix} \ . \tag{35}$$

Such a system can easily be solved in a stepwise fashion starting with the last equation, which provides

$$x_N = \frac{b'_N}{a'_{NN}} \ , \tag{36}$$

and then successively working one's way up to the first equation using the solutions from the previous equations as input, i.e.,

$$x_k = \frac{1}{a'_{kk}} \left(b'_k - \sum_{i=k+1}^{N} a'_{ki} x_i \right) \ , \qquad \text{for} \qquad k = N-1, \ldots, 1 \ , \tag{37}$$

where all the x_i's on the right-hand side are known from previous solution steps. This procedure is called *backward substitution*.

Note that for the system of equations (33) to have a solution, this requires

$$\det \mathsf{A} \neq 0 \ , \tag{38}$$

and since[1]

$$\det \mathsf{A} = (-1)^S \prod_{i=1}^{N} a'_{ii} \ , \tag{39}$$

where S is the total number of row swaps during the triangularization process as explained below,[2] this implies that all elements on the diagonal are non-zero, i.e., $a_{ii} \neq 0$ for all $i = 1, 2, \ldots, N$. This ensures that the divisions by diagonal elements in eqs. (36) and (37) are well defined.

To achieve the triangular structure, we make use of the fact that in a system of linear equations like (32), we may subtract arbitrary multiples of one equation from another equation without changing the solutions x_i. Since subtractions of this kind affect both the original coefficients a_{ij} and the right-hand side values b_i, we take the original coefficient matrix A and append the vector $\mathbf{b}$ to form an $N \times (N+1)$ matrix

$$\tilde{\mathsf{A}} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1N} & b_1 \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2N} & b_2 \\ a_{31} & a_{32} & a_{33} & \ldots & a_{2N} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \ldots & a_{NN} & b_N \end{pmatrix} \ , \tag{40}$$

also referred to as the *augmented matrix*. The manipulation of the original system of equations $\mathsf{A}\mathbf{x} = \mathbf{b}$ can now be understood as subtracting multiples of one row from another row of this matrix (which will not change the determinant of A as long as we do not change any row by an overall factor).

We start by eliminating all elements from the first column *except* for the first element a_{11}. This is easily achieved for the n-th element a_{n1} ($n = 2, 3, \ldots, N$) by subtracting from the n-th row the first row multiplied by a_{n1}/a_{11}. Note here that since $\det \mathsf{A} \neq 0$, at least one of the elements a_{i1} ($i = 1, 2, \ldots, N$) in the first column must be non-zero. It is possible, however, that a_{11} itself might be zero. In this case, we could not use the first row to eliminate the first-column coefficients of

[1] While the triangularization of A does not change the value of the determinant, this particular expression is of course only true for the triangularized version.

[2] Starting with $S = 0$, S is advanced by 1 every time two rows are swapped, i.e., S is an integer between 0 and $N - 1$.

the subsequent rows. To take care of this possibility, we will proceed in the following manner: We search for the element a_{j1} in the first column that is *largest* in magnitude. We then simply swap the rows j and 1 (including the elements $b_j \leftrightarrow b_1$ in the last column), which corresponds to a renumbering of the equations, but will not change the numbering for the unknowns x_i. This achieves two things. First, we ensure that the new a'_{11} is non-zero *and*, second, that all necessary divisions by a'_{11} are done with the largest available denominator which will minimize numerical round-off error. This procedure of swapping rows in this manner is called *pivoting*.

Hence, after pivoting the first row (which produced new elements a'_{1j} for the first row[3]), we subtract multiples of the first row from each subsequent row according to

$$a_{ij} \to a'_{ij} = a_{ij} - \frac{a_{i1}}{a'_{11}} a'_{1j} \ , \qquad \text{for} \qquad i = 2, 3, \ldots, N \quad \text{and} \quad j = 1, 2, 3, \ldots, N+1 \ , \tag{41}$$

where $a_{i(N+1)} \equiv b_i$. This produces

$$\tilde{\mathsf{A}} \to \tilde{\mathsf{A}}_1 = \left(\begin{array}{c|cccc} a'_{11} & a'_{12} & a'_{13} & \ldots & a'_{1N} & b'_1 \\ \hline 0 & a'_{22} & a'_{23} & \ldots & a'_{2N} & b'_2 \\ 0 & a'_{32} & a'_{33} & \ldots & a'_{2N} & b'_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{N2} & a'_{N3} & \ldots & a'_{NN} & b'_N \end{array} \right) \ , \tag{42}$$

where the auxiliary vertical and horizontal lines are drawn to guide the eye.

We then proceed in exactly the same manner for the remaining $(N-1)\times N$-dimensional submatrix in the lower right-hand corner of (42), i.e., we first pivot the rows so that the new element a'_{22} is largest in magnitude in the first column of the submatrix and we eliminate all other elements in this column by subtraction analogous to (41) thus producing a new $(N-2)\times(N-1)$-dimensional submatrix that can then again be processed as before.

Hence, in general, assuming that pivoting is done before each elimination step, the k-th step corresponds to

$$a_{ij} \to a'_{ij} = a_{ij} - \frac{a_{ik}}{a'_{kk}} a'_{kj} \ , \qquad \text{for} \qquad i = k+1, k+2, \ldots, N \quad \text{and} \quad j = 1, 2, \ldots, N+1 \ , \tag{43}$$

where $k = 1, 2, \ldots, N$. This equation, together with the pivoting prescription and the backward substitution of eqs. (36) and (37), provides a complete description of the Gauss–Jordan elimination scheme.

6 To-Do List

- All numerical work needs to be done in double precision (real numbers 15 digits of accuracy).
- If at all possible, code should be modular, i.e., independent of the specific problem and reusable for other purposes.
- Build error checking into the codes. Test your codes. When doing so, make sure that you choose *non-trivial* examples.

[3]For simplicity, I continue to use unprimed elements for all other rows even though the pivot-swapped j-th row is different from the original.

1. Write a routine that performs Gauss–Jordan elimination with pivoting for a given augmented matrix of dimension $N \times (N+1)$. The routine should return the solution vector $\mathbf{x}$ and the value of the determinant. The routine need not preserve the original coefficient matrix. (Write the routine yourself. Do *not* use a canned routine.)

2. Write a routine that performs Gaussian integration for an arbitrary function provided by a user-supplied function routine. The integration routine should allow for arbitrary mesh sizes and transformation functions. You may input the Gaussian mesh points and weights via a data file or use any one of the routines available in the literature or online to calculate them (see Appendix for a FORTRAN subroutine adapted from the *Numerical Recipes* [5]). If you do the latter, it might be better to run this code for a number of desired mesh sizes (for example, $N = 10, 12, 14, \ldots, 48$) and store the results on data files for repeated use later, instead of running the code every time you need a mesh.

3. Write a routine that generates the potential matrix elements $U(p, q)$ for a given mesh. In addition you need $U(p, k)$ and $U(p, \beta)$ for parameters k and β that, in general, will be different from any mesh point.

4. Write a routine that prepares the coefficient matrix and the inhomogeneity vector of the integral equation, i.e., you need to generate the augmented matrix for the Gauss–Jordan elimination.

5. Solve the system of linear equations and calculate the Jost function. Depending on whether one treats the scattering or the bound-state case, calculate phase shifts and scattering lengths or the binding energy. Plot phase shifts (in degrees vs. energy in MeV). The binding-energy calculation requires searching for the root(s) of $F_\beta(-\alpha^2)$.

Final Report due: TBD

Appendix: FORTRAN Code for Gaussian Mesh Points and Weights

The following FORTRAN code adapted from the *Numerical Recipes* [5] generates N mesh points x_i and weights w_i for Gaussian integrations.

`x1`	Lower bound of integration (input)
`x2`	Upper bound of integration (input)
`x`	Array with mesh points (output)
`w`	Array with weights (output)
`n`	Grid size (input)

All variables are double precision. For finite integration bounds, the lower and upper limits `x1` and `x2`, respectively, of the integration can be put in explicitly. For infinite bounds (as is the case here), one chooses `x1` $= -$`1.d0` and `x2` $= +$`1.d0` to obtain the grid for the subsequent transformation, as explained in sec. 5.1. The calling program needs to provide storage for the double-precision arrays `x` and `w`.

```
      subroutine gauleg(x1,x2,x,w,n)
      implicit real*8 (a-h,o-z)
      dimension x(n),w(n)
      parameter (eps=3.d-14)
      pi = 4.d0*datan(1.0d0)
      m=(n+1)/2
      xm=0.5d0*(x2+x1)
      xl=0.5d0*(x2-x1)
      do i=1,m
        z=dcos(pi*(i-.25d0)/(n+.5d0))
  100   continue
          p1=1.d0
          p2=0.d0
          do j=1,n
            p3=p2
            p2=p1
            p1=((2.d0*j-1.d0)*z*p2-(j-1.d0)*p3)/j
          enddo
          pp=n*(z*p1-p2)/(z*z-1.d0)
          z1=z
          z=z1-p1/pp
        if (dabs(z-z1).gt.eps) goto 100
        x(i)=xm-xl*z
        x(n+1-i)=xm+xl*z
        w(i)=2.d0*xl/((1.d0-z*z)*pp*pp)
        w(n+1-i)=w(i)
      enddo
      return
      end
```

References

[1] H. Haberzettl, *Quantum Mechanics — Lecture Notes*.

[2] E.A. Bartnik, H. Haberzettl, and W. Sandhas, *Two-body bound state problem and nonsingular scattering equations*, Phys. Rev. C **34**, 1520-1529 (1986).

[3] R.A. Malfliet and J.A. Tjon, Nucl. Phys. **A127**, 161 (1969). (Note: The MT I&III parameters in the table of sec. 3 differ slightly from the original ones; use the parameters given here.)

[4] M. Abramowitz and I.A. Stegun, *Handbook of Mathematical Functions* (Dover, 1972).

[5] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes — The Art of Scientific Computing*, Cambridge University Press (1986). (For newer editions, see `http://www.nr.com/`.)