993SM - Laboratory of Computational Physics II week October 4, 2024

Maria Peressi

Università degli Studi di Trieste - Dipartimento di Fisica

Sede di Miramare (Strada Costiera 11, Trieste)

e-mail: peressi@units.it

tel.: +39 040 2240242

more on the Numerov's algorithm

- deeper analysis of the Numerov's algorithm
- numerical analysis: search for zeros

1D Schroedinger equation
and the Numerov's method

The Scroedinger eq.:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = \overset{\text{unknown}}{\overset{\downarrow}{E}}\overset{\text{unknown}}{\overset{\swarrow}{\psi}}(x);$$

has the form:

$$\boxed{\frac{d^2y}{dx^2} = -g(x)y(x) + s(x)}$$

with:

$$\boxed{g(x) = (2m/\hbar^2)[E - V(x)] \text{ and } s(x) = 0}.$$

1D Schroedinger equation for harmonic oscillator

and the Numerov's method

Since $s(x) = 0$, the Numerov's formula reduces to :

$$y_{n+1}\left[1 + g_{n+1}\frac{(\Delta x)^2}{12}\right] = 2y_n\left[1 - 5g_n\frac{(\Delta x)^2}{12}\right] - y_{n-1}\left[1 + g_{n-1}\frac{(\Delta x)^2}{12}\right] + O[(\Delta x)^6]$$

Defining: $f_n \equiv 1 + g_n\dfrac{(\Delta x)^2}{12}$ with $g(x) = 2\left(\boxed{\epsilon} - \dfrac{x^2}{2}\right)$

we rewrite Numerov's formula as

$$\boxed{y_{n+1} = \frac{(12 - 10f_n)y_n - f_{n-1}y_{n-1}}{f_{n+1}}}$$

contain ε

inside the Numerov's method

Find a solution ψ with a given number `n` of nodes and energy `En`

Possibility 1): we fix `En`

Possibility 2): we do not know `En`, but we start from an initial energy guess `E`; the guess is on an energy interval `[Elow, Eup]` that we know for sure to contain `En`

For each value of E: start integrating ψ from `x=0` towards positive values of `x` (forward)

During integration, count for the number `ncross` of changes of sign of ψ:

if `ncross>n` => `E` is too high => choose `[Elow, E]`

if `ncross<n` => `E` is too low => choose `[E, Eup]`

inside the Numerov's method

Find a solution ψ with a given number `n` of nodes and energy `En`

Possibility 1): we fix `En`

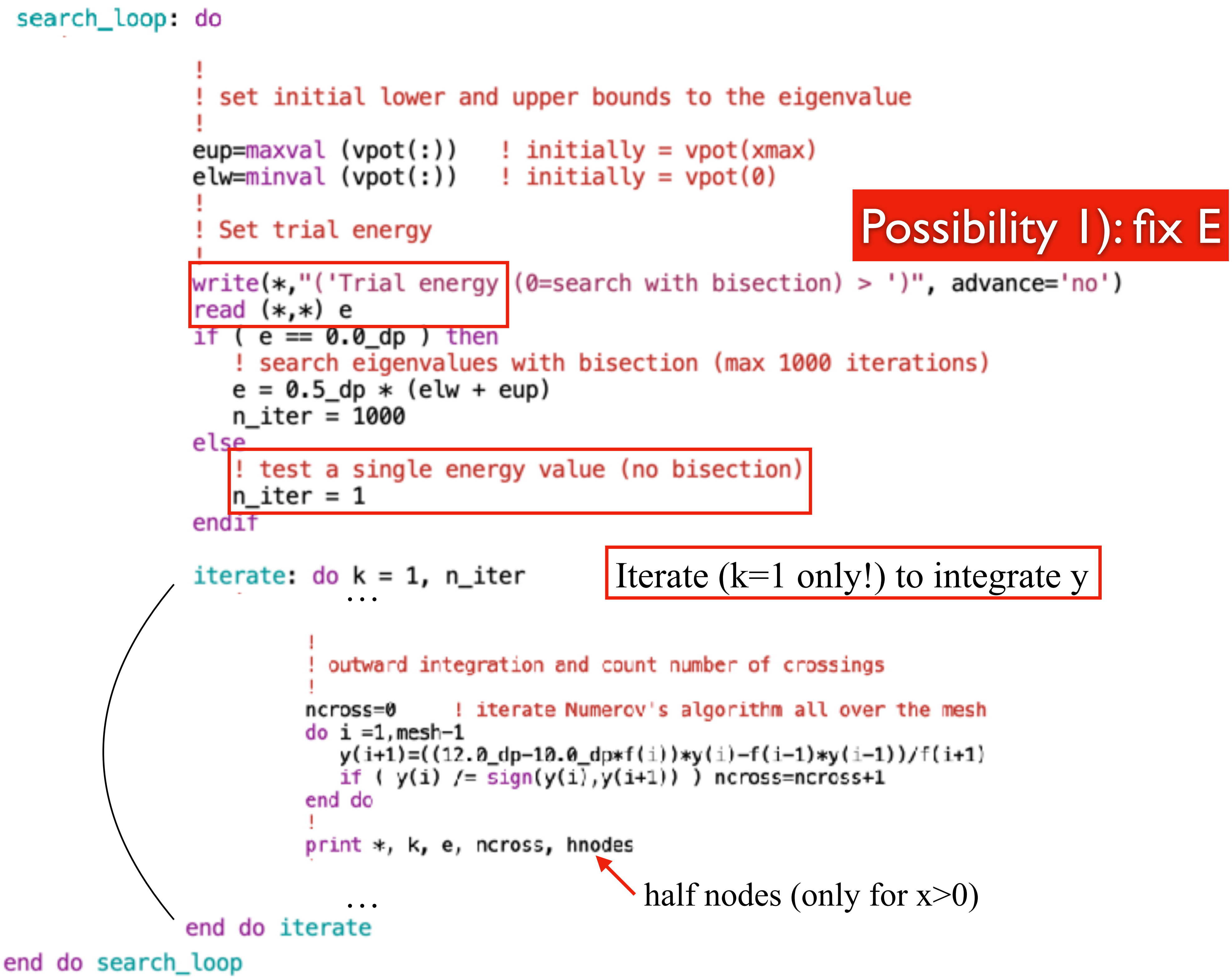
Possibility 2): we do not know `En`, but we start from an initial energy guess `E`; the guess is on an energy interval `[Elow, Eup]` that we know for sure to contain `En`

For each value of E: start integrating ψ from `x=0` towards positive values of `x` (forward)

During integration, count for the number `ncross` of changes of sign of ψ:

if `ncross>n` => `E` is too high => choose `[Elow, E]`

if `ncross<n` => `E` is too low => choose `[E, Eup]`



Possibility 1): fix E

```
search_loop: do

           !
           ! set initial lower and upper bounds to the eigenvalue
           !
           eup=maxval (vpot(:))    ! initially = vpot(xmax)
           elw=minval (vpot(:))    ! initially = vpot(0)
           !
           ! Set trial energy
           !
           write(*,"('Trial energy (0=search with bisection) > ')", advance='no')
           read (*,*) e
           if ( e == 0.0_dp ) then
              ! search eigenvalues with bisection (max 1000 iterations)
              e = 0.5_dp * (elw + eup)
              n_iter = 1000
           else
              ! test a single energy value (no bisection)
              n_iter = 1
           endif

           iterate: do k = 1, n_iter
                  …

                 !
                 ! outward integration and count number of crossings
                 !
                 ncross=0      ! iterate Numerov's algorithm all over the mesh
                 do i =1,mesh-1
                    y(i+1)=((12.0_dp-10.0_dp*f(i))*y(i)-f(i-1)*y(i-1))/f(i+1)
                    if ( y(i) /= sign(y(i),y(i+1)) ) ncross=ncross+1
                 end do
                 !
                 print *, k, e, ncross, hnodes

                  …
          end do iterate
end do search_loop
```

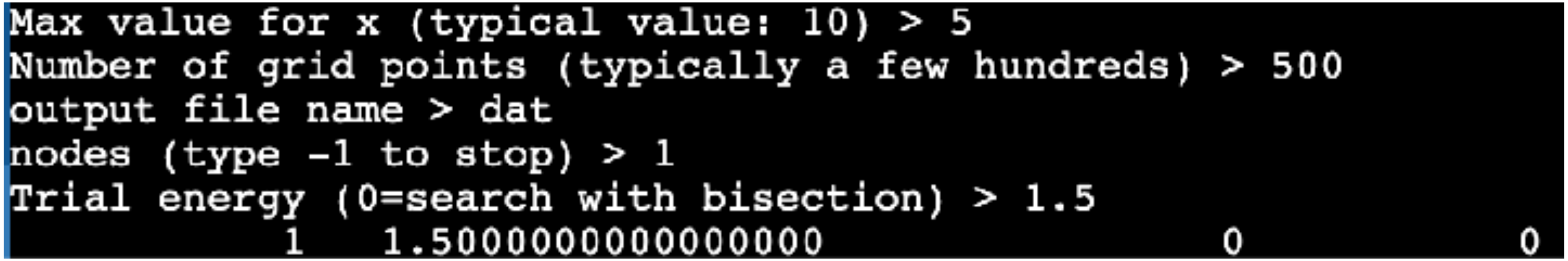
Iterate (k=1 only!) to integrate y

half nodes (only for x>0)

0. (ADD!) Calculate and plot eigenfunctions for various values of "**nodes**" (number of **nodes** of the eigenfunction), from nodes=0, 1, 2,… Input also the precise energy value, which is (nodes+1/2).

It may be useful to plot, together with eigenfunctions or eigenfunctions squared, the classical probability density, contained in the fourth column of the output file. It will clearly show the classical inversion points. See **harmonic0.gpl** on Moodle as an example of a macro for **gnuplot**.

I/O on the screen:



```
Max value for x (typical value: 10) > 5
Number of grid points (typically a few hundreds) > 500
output file name > dat
nodes (type -1 to stop) > 1
Trial energy (0=search with bisection) > 1.5
          1   1.5000000000000000                  0                  0
```

are equal, ok!

the last line comes from this command in the code:

```
print *, k, e, ncross, hnodes
```

number of iterations

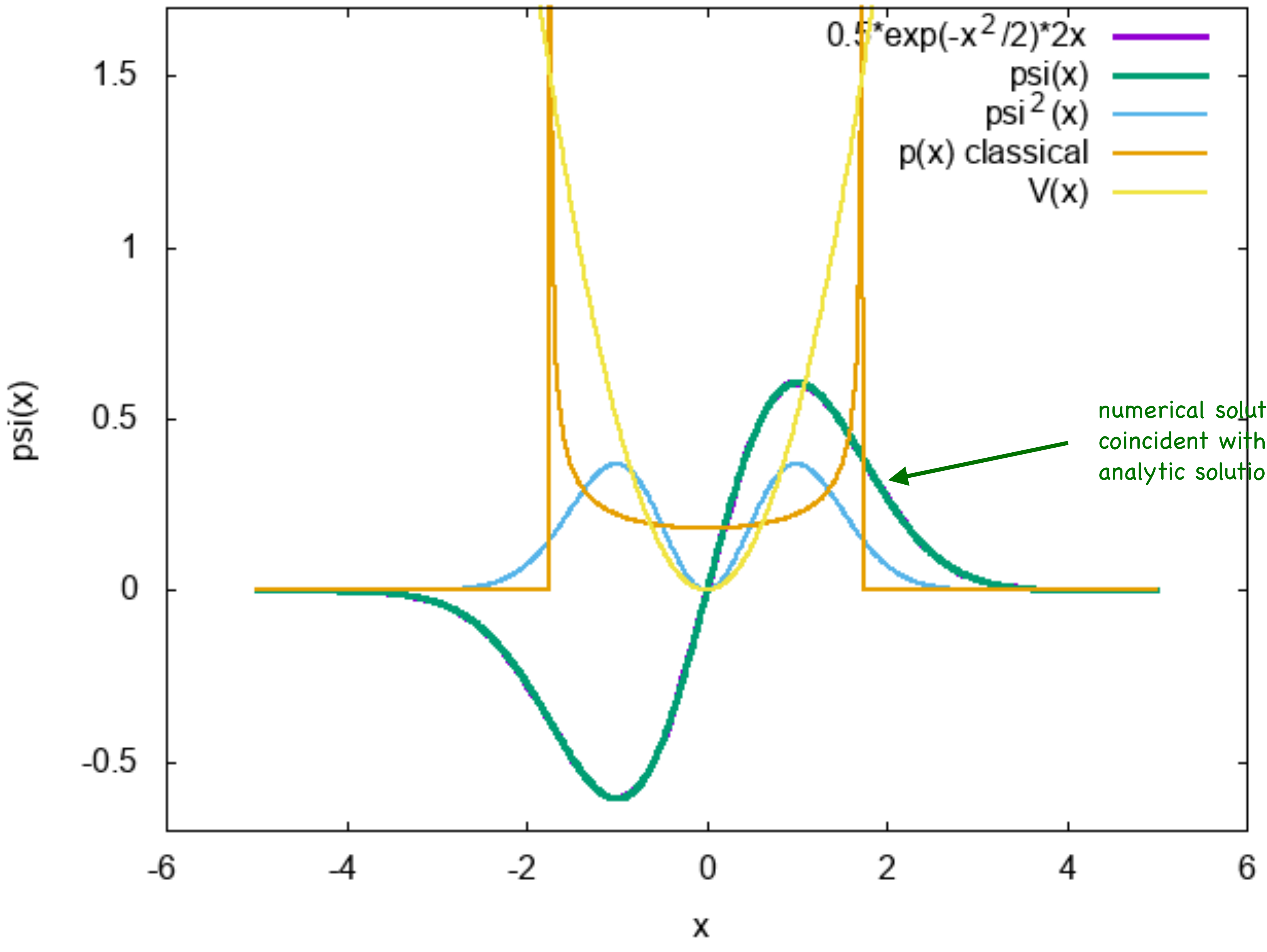
energy

number of nodes for x>0 numerically found

number of nodes for x>0 expected:
hnodes = nodes/2
(within the integers!)

Solution for n=1

and energy exact (1.5)



numerical solution coincident with the exact analytic solution

inside the Numerov's method

Find a solution ψ with a given number `n` of nodes and energy `En`

Possibility 1): we fix `En`

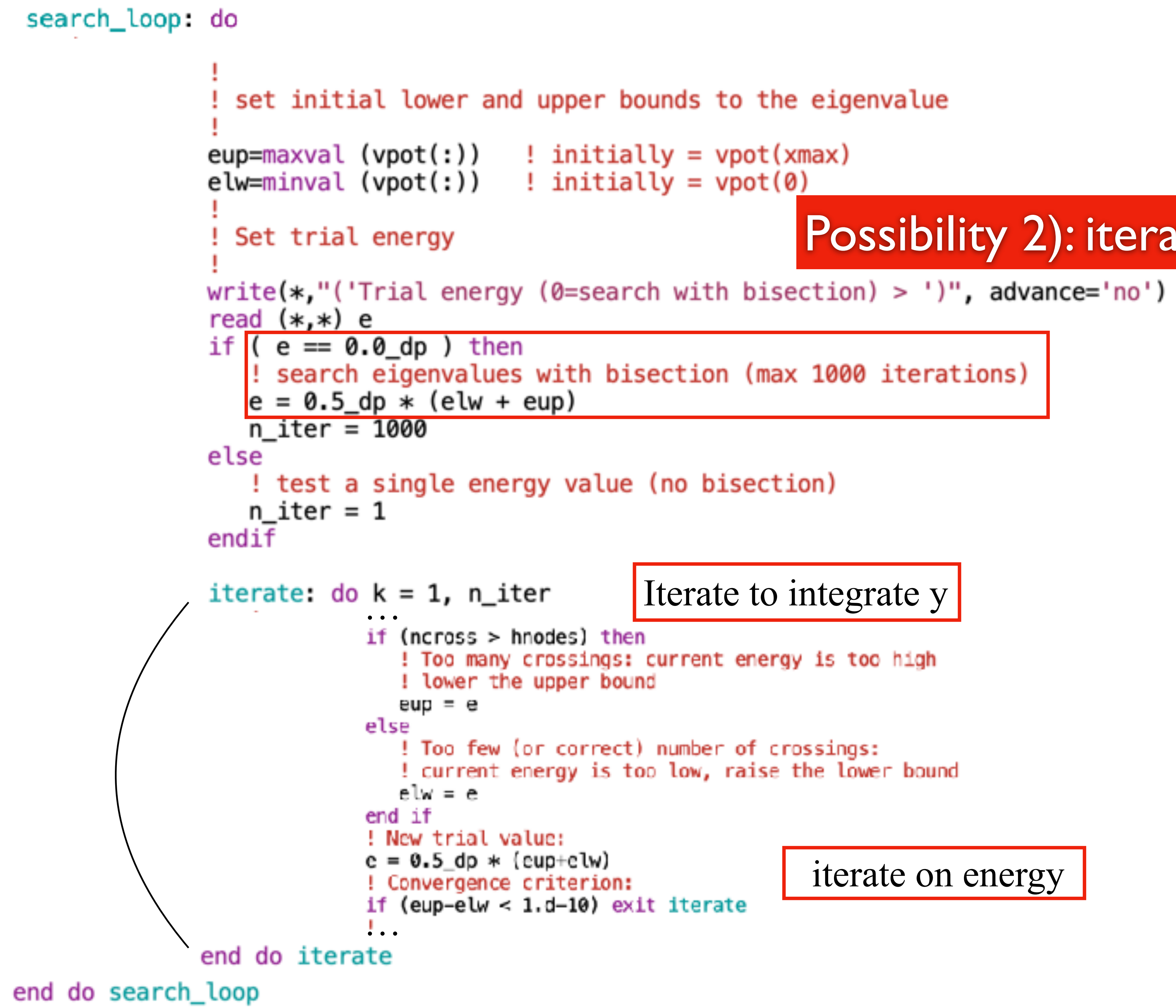
Possibility 2): we do not know `En`, but we start from an initial energy guess `E`; the guess is on an energy interval `[Elow, Eup]` that we know for sure to contain `En`

For each value of E: start integrating ψ from `x=0` towards positive values of `x` (forward)

During integration, count for the number `ncross` of changes of sign of ψ:
if `ncross>n` => `E` is too high => choose `[Elow, E]`
if `ncross<n` => `E` is too low => choose `[E, Eup]`

We expect a set of orthogonal eigenvectors with increasing number of zeros for increasing energy eigenvalue; more precisely, equal to the energy quantum number n, because the number of zeros is equal to the degree of the polynomial

Possibility 2): iterate on E



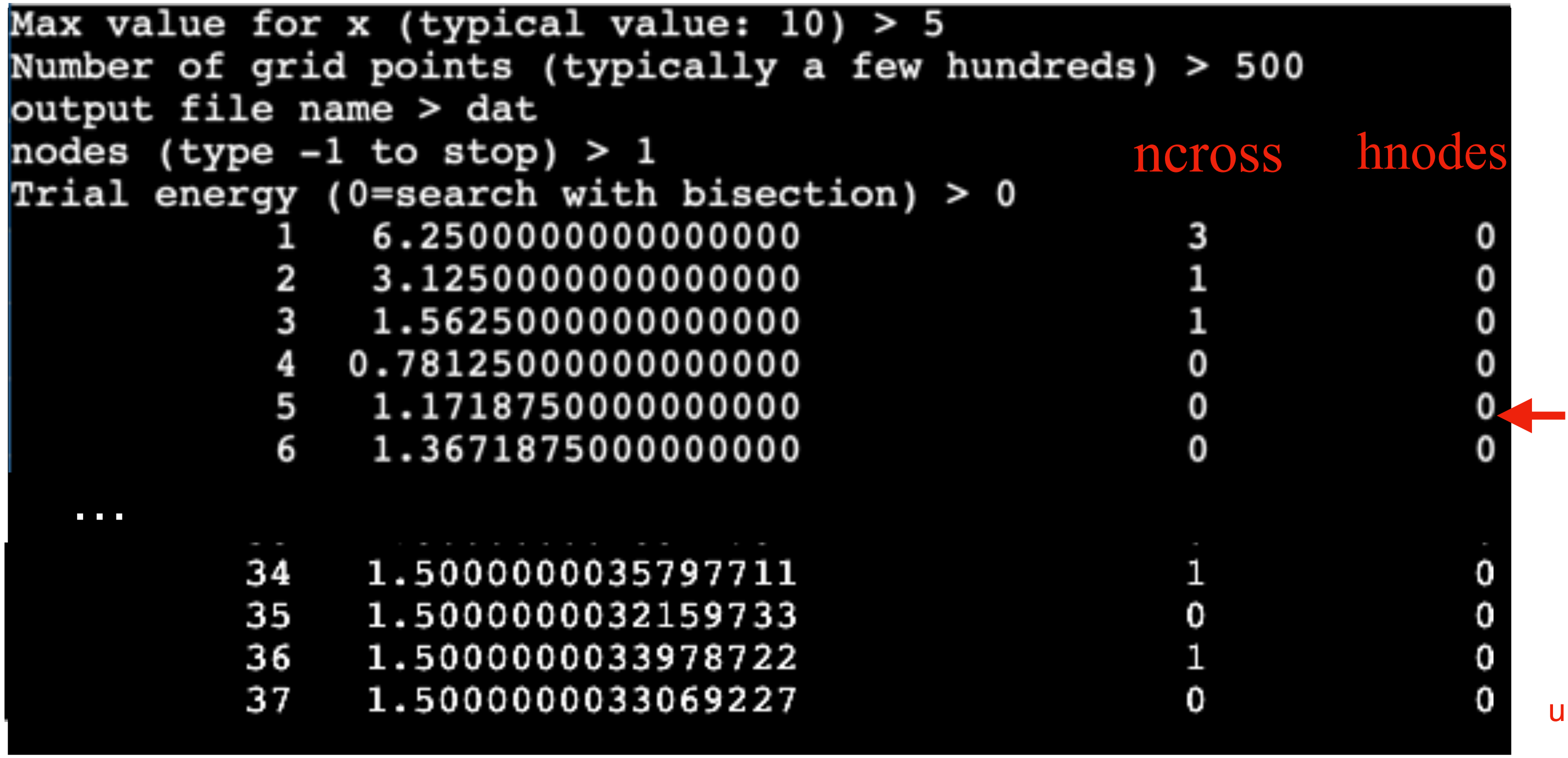
```
search_loop: do

           !
           ! set initial lower and upper bounds to the eigenvalue
           !
           eup=maxval (vpot(:))    ! initially = vpot(xmax)
           elw=minval (vpot(:))    ! initially = vpot(0)
           !
           ! Set trial energy
           !
           write(*,"('Trial energy (0=search with bisection) > ')", advance='no')
           read (*,*) e
           if ( e == 0.0_dp ) then
              ! search eigenvalues with bisection (max 1000 iterations)
              e = 0.5_dp * (elw + eup)
              n_iter = 1000
           else
              ! test a single energy value (no bisection)
              n_iter = 1
           endif

           iterate: do k = 1, n_iter
                       ...
                       if (ncross > hnodes) then
                          ! Too many crossings: current energy is too high
                          ! lower the upper bound
                          eup = e
                       else
                          ! Too few (or correct) number of crossings:
                          ! current energy is too low, raise the lower bound
                          elw = e
                       end if
                       ! New trial value:
                       e = 0.5_dp * (eup+elw)
                       ! Convergence criterion:
                       if (eup-elw < 1.d-10) exit iterate
                       !...
           end do iterate
end do search_loop
```

Iterate to integrate y

iterate on energy

1. Calculate and plot eigenfunctions for various values of n (number of nodes of the eigenfunction), from n=0, 1, 2,... Let the code find the correct energy value (choosing "0" when the code asks for an energy value). It may be useful to plot, together with eigenfunctions or eigenfunctions squared, the classical probability density, contained in the fourth column of the output file. It will clearly show the classical inversion points. See `harmonic0.gpl` on Moodle as an example of a macro for `gnuplot`.

I/O on the screen:



```
Max value for x (typical value: 10) > 5
Number of grid points (typically a few hundreds) > 500
output file name > dat
nodes (type -1 to stop) > 1
Trial energy (0=search with bisection) > 0
           1    6.2500000000000000          3           0
           2    3.1250000000000000          1           0
           3    1.5625000000000000          1           0
           4   0.78125000000000000          0           0
           5    1.1718750000000000          0           0
           6    1.3671875000000000          0           0
  ...
          34    1.5000000035797711          1           0
          35    1.5000000032159733          0           0
          36    1.5000000033978722          1           0
          37    1.5000000033069227          0           0
```

ncross

hnodes

ncross=hnodes is ok,
but the stop criterion
is a tolerance on the energy
(now the uncertainty is ~0.4)

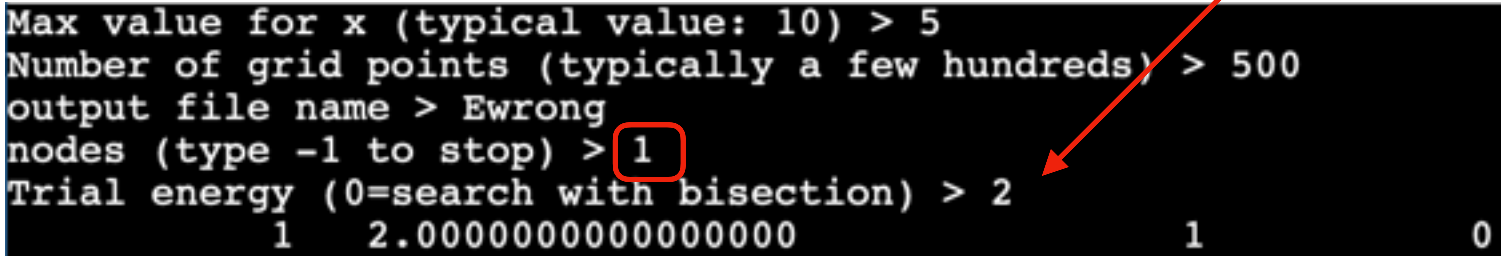
uncertainty < tolerance = 10^-10

The results

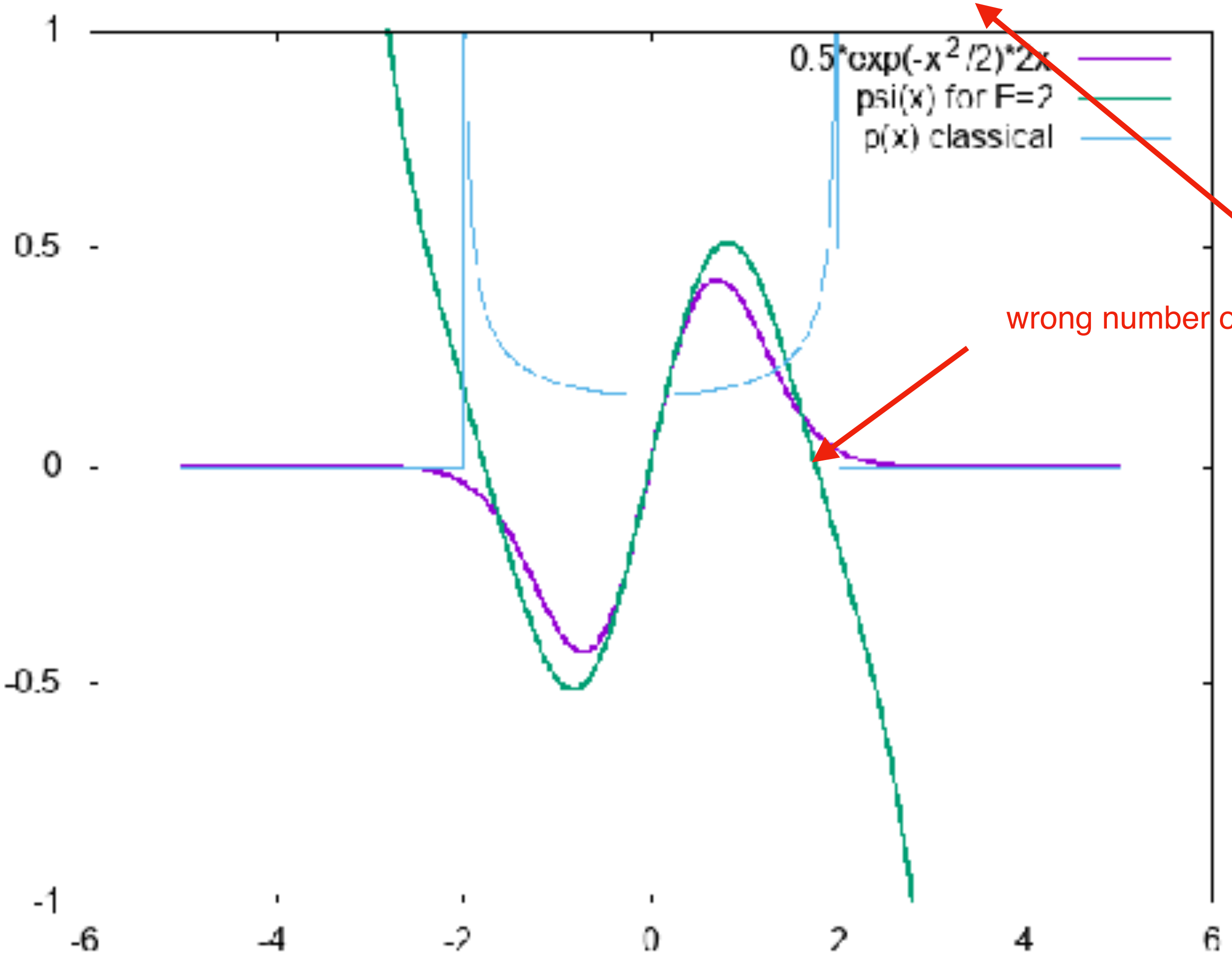
2. Specify an energy value not corresponding to an eigenvalue. Look at the resulting wavefunctions.

1.5 < 2 < 2.5

I/O on the screen:



```
Max value for x (typical value: 10) > 5
Number of grid points (typically a few hundreds) > 500
output file name > Ewrong
nodes (type -1 to stop) > 1
Trial energy (0=search with bisection) > 2
           1   2.0000000000000000                    1           0
```

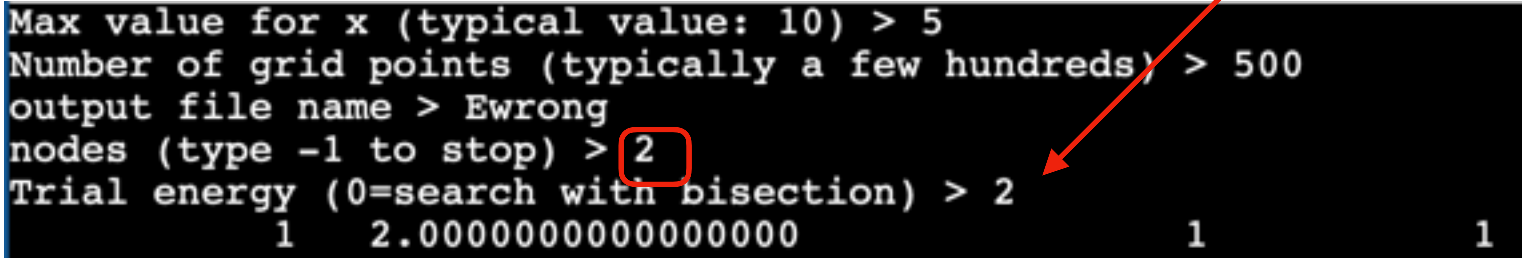


wrong number of nodes!

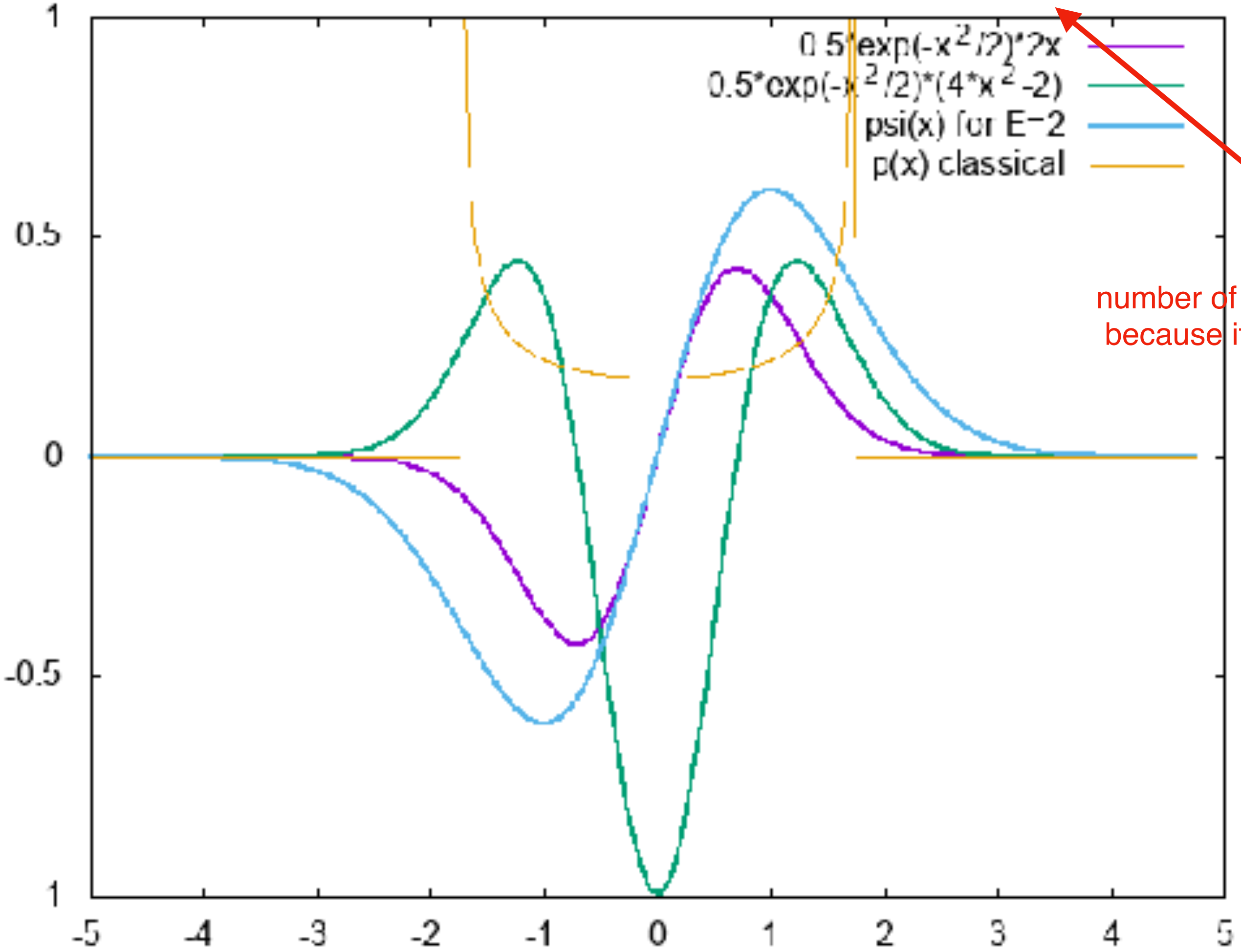
2. Specify an energy value not corresponding to an eigenvalue. Look at the resulting wavefunctions.

1.5 < 2 < 2.5

I/O on the screen:



```
Max value for x (typical value: 10) > 5
Number of grid points (typically a few hundreds) > 500
output file name > Ewrong
nodes (type -1 to stop) > 2
Trial energy (0=search with bisection) > 2
           1   2.0000000000000000                        1            1
```

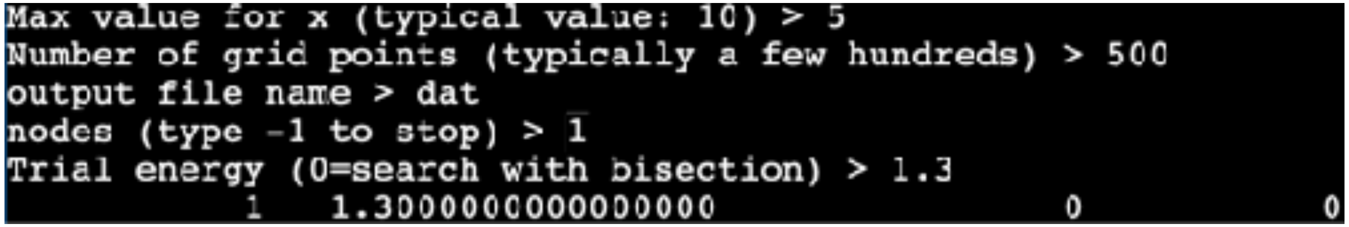


number of nodes apparently correct because it disregards the central one!

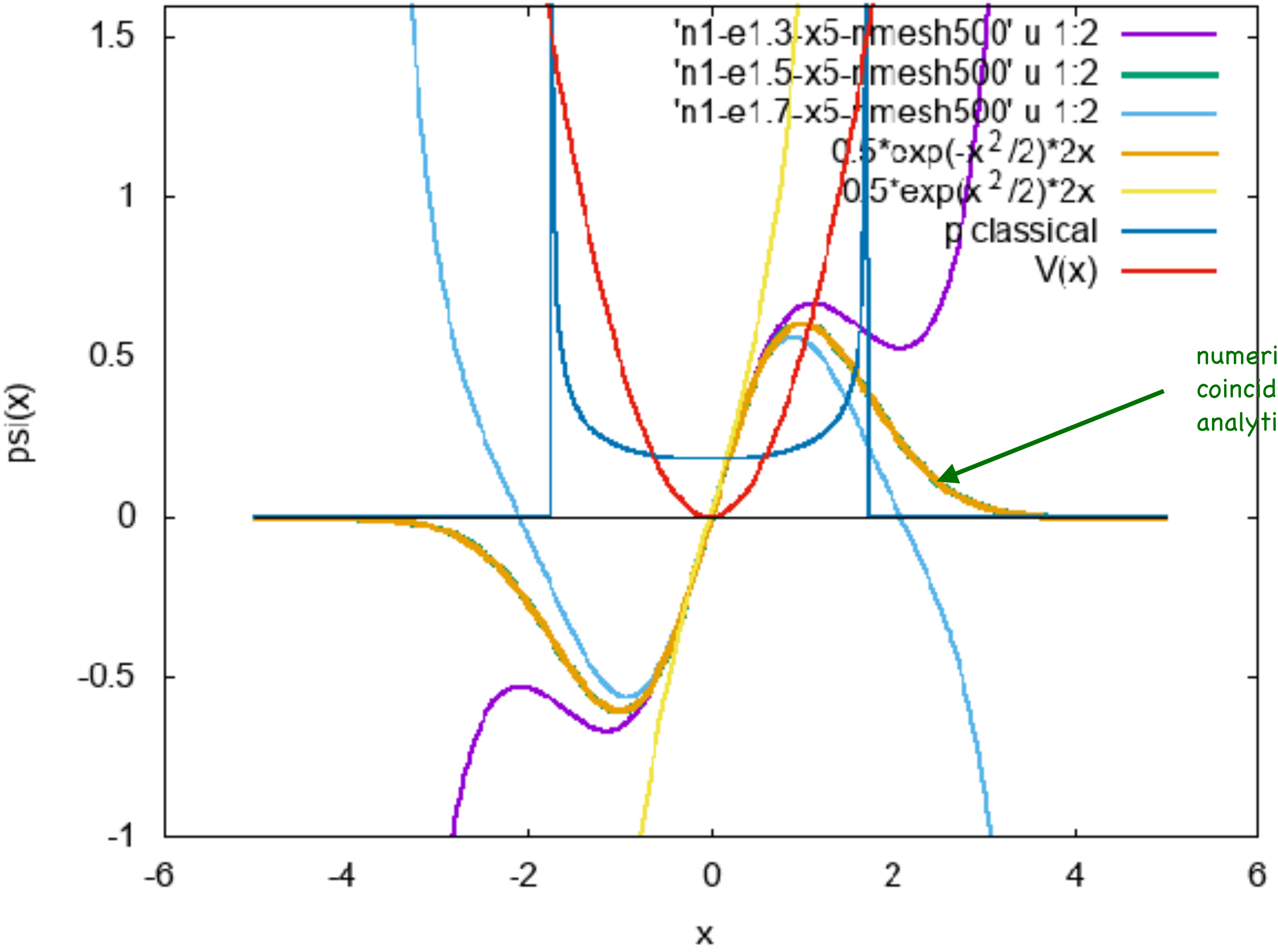
???

solution apparently reasonable? but closer to H_1 rather than H_2

3. Specify an energy value close to but not exactly corresponding to an eigenvalue. Look at the resulting wavefunctions.



```
Max value for x (typical value: 10) > 5
Number of grid points (typically a few hundreds) > 500
output file name > dat
nodes (type -1 to stop) > 1
Trial energy (0=search with bisection) > 1.3
           1   1.3000000000000000                    0              0
```

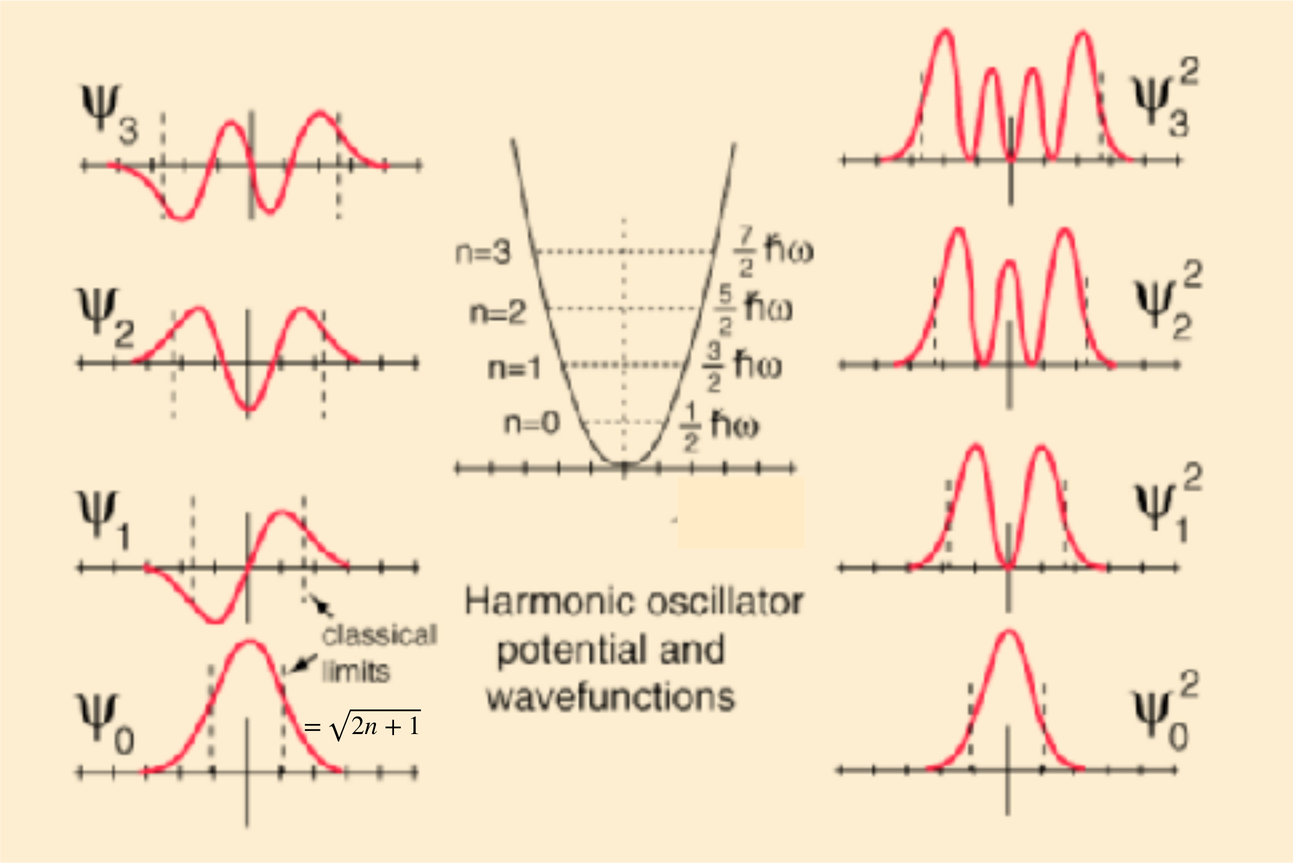


numerical solution coincident with the exact analytic solution

4. Examine the effects of the parameters `xmax`, `mesh`. For a given Δx, how large can be the number of nodes?

Hint: the nodes are internal to the classical limit range =>

$$\sqrt{2n+1} < \mathtt{xmesh} \quad \Longrightarrow \quad n_{max} < (\mathtt{xmesh}^2 - 1)/2$$



inside the Numerov's method

Used to count the number of nodes while “building” the wavefunction:

```
if ( y(i) /= sign(y(i),y(i+1)) ) ncross=ncross+1
```

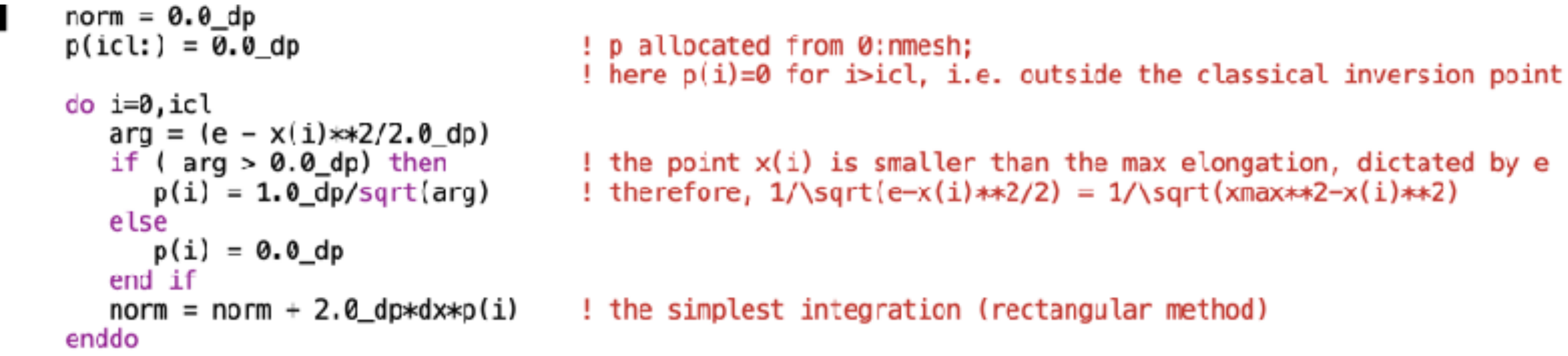
(`ncross` means crossing with the x axis, i.e., zeros)

Used to count the number of change of sign of 1-f, determining the classical inversion point

```
f(i)=ddx12*(2.0_dp*(vpot(i)-e))
if ( f(i) /= sign(f(i),f(i-1)) ) icl=i
```

possibility 2):
guess on the energy,
then refine
(iterating, “shooting” method)

inside the Numerov's method: (rectangular) numerical integration



```
norm = 0.0_dp
p(icl:) = 0.0_dp                   ! p allocated from 0:nmesh;
                                   ! here p(i)=0 for i>icl, i.e. outside the classical inversion point

do i=0,icl
   arg = (e - x(i)**2/2.0_dp)
   if ( arg > 0.0_dp) then         ! the point x(i) is smaller than the max elongation, dictated by e
      p(i) = 1.0_dp/sqrt(arg)      ! therefore, 1/\sqrt(e-x(i)**2/2) = 1/\sqrt(xmax**2-x(i)**2)
   else
      p(i) = 0.0_dp
   end if
   norm = norm + 2.0_dp*dx*p(i)    ! the simplest integration (rectangular method)
enddo
```

Polinomi ortogonali classici rispetto al peso w nell'intervallo I:

$$\int_I w(x)P_n(x)P_m(x) \propto \delta_{n,m}$$

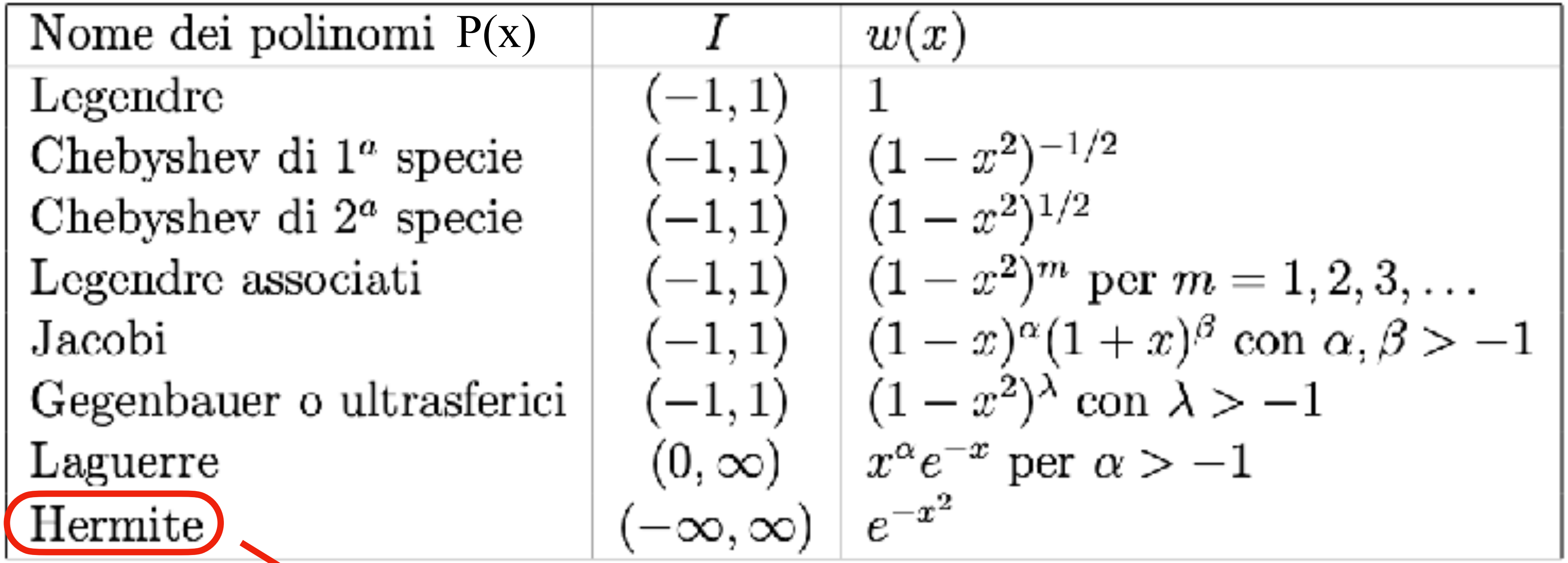
Nome dei polinomi P(x)	I	$w(x)$
Legendre	$(-1,1)$	1
Chebyshev di 1^a specie	$(-1,1)$	$(1-x^2)^{-1/2}$
Chebyshev di 2^a specie	$(-1,1)$	$(1-x^2)^{1/2}$
Legendre associati	$(-1,1)$	$(1-x^2)^m$ per $m=1,2,3,\ldots$
Jacobi	$(-1,1)$	$(1-x)^\alpha(1+x)^\beta$ con $\alpha,\beta>-1$
Gegenbauer o ultrasferici	$(-1,1)$	$(1-x^2)^\lambda$ con $\lambda>-1$
Laguerre	$(0,\infty)$	$x^\alpha e^{-x}$ per $\alpha>-1$
Hermite	$(-\infty,\infty)$	e^{-x^2}

$$\int_{-\infty}^{\infty} dx\, e^{-x^2} H_n(x)H_m(x) = \sqrt{\pi}2^n n!\delta_{n,m}$$

$$\begin{aligned}
H_0 &= 1\\
H_1 &= 2x\\
H_2 &= 4x^2-2\\
H_3 &= 8x^3-12x\\
H_4 &= 16x^4-48x^2+12
\end{aligned}$$

Polinomi ortogonali classici rispetto al peso w nell'intervallo I:

$$\int_I w(x)P_n(x)P_m(x) \propto \delta_{n,m}$$



Nome dei polinomi P(x)	I	$w(x)$
Legendre	$(-1,1)$	1
Chebyshev di 1^a specie	$(-1,1)$	$(1-x^2)^{-1/2}$
Chebyshev di 2^a specie	$(-1,1)$	$(1-x^2)^{1/2}$
Legendre associati	$(-1,1)$	$(1-x^2)^m$ per $m=1,2,3,\dots$
Jacobi	$(-1,1)$	$(1-x)^\alpha(1+x)^\beta$ con $\alpha,\beta>-1$
Gegenbauer o ultrasferici	$(-1,1)$	$(1-x^2)^\lambda$ con $\lambda>-1$
Laguerre	$(0,\infty)$	$x^\alpha e^{-x}$ per $\alpha>-1$
Hermite	$(-\infty,\infty)$	e^{-x^2}

$$\boxed{\int_{-\infty}^{\infty} dx\, e^{-x^2} H_n(x)H_m(x) = \sqrt{\pi}2^n n!\delta_{n,m}}$$

$$\begin{aligned} H_0 &= 1 \\ H_1 &= 2x \\ H_2 &= 4x^2-2 \\ H_3 &= 8x^3-12x \\ H_4 &= 16x^4-48x^2+12 \end{aligned}$$

Part II: search for zeros

searching for zeros of a function
x_0=? bisection method

1. determine an interval $[x_L,x_U]$ at whose extremes the function $y(x)$ has discordant signs (to be sure that it contains x_0)
2. calculate the midpoint of the interval $[x_L,x_U]$: $x_M = (x_L + x_U)/2$ and evaluate $y(x_M)$;
3. If $y(x_M)=0$ then $x_M = x_0$ and the search ends.
4. Otherwise, take as the new interval the one at whose extremities the function has discordant signs (depending on the case it will be necessary to redefine $x_L=x_M$ or $x_U=x_M$): it contains x_0
5. iterate points 2 - 4 until:

a) the uncertainty on the location of x_0 decreases below a pre-established an absolute threshold ($|x_U-x_L| < \varepsilon$), or a relative threshold ($|x_U-x_L| < \varepsilon\, |x_L|$ or $< \varepsilon\, |x_U|$, where x_0 is approximated by x_L or x_U); or

b) $|y(x_M)| < \varepsilon'$; or

c) a maximum number of iterations is exceeded.

searching for zeros of a function x_0=? bisection method

1. determine an interval [x_L,x_U] at whose extremes the function y(x) has discordant signs (to be sure that it contains x_0)

Implementation:

```
y(xL)* y(xU) < 0
```

```
y(xL)/= sign(y(xL),y(xU)) or y(xU)/= sign(y(xU),y(xL))
```

searching for zeros of a function x_0=? bisection method

4. take as the new interval the one at whose extremities the function has discordant signs (depending on the case it will be necessary to redefine x_L=x_M or x_U=x_M): it contains x_0

Implementation:

```
if (y(xL)*y(xM) < 0) then
    xU = xM
else if (y(xU)*y(xM) < 0) then
    xL = xM
end if
```

searching for zeros of a function
x_0=? bisection method

5. iterate points 2 - 4 until:

a) the uncertainty on the location of x_0 decreases below a pre-established an absolute threshold ($|x_U - x_L| < \varepsilon$), or a relative threshold ($|x_U - x_L| < \varepsilon\,|x_L|$ or $< \varepsilon\,|x_U|$, where x_0 is approximated by x_L or x_U)

Implementation - which is the best criterion?

- use an absolute threshold: possible problems (for roundoff errors) if x_0 is large
- use a relative threshold: possible problems if x_0 is small
- other possible problems if y(x) is too flat close to x_0

Exercise:

implement this algorithm
if you never did,
or if you have forgotten...