Numerov algorithm

The radial equation is usually solved with Numerov algorithm which is designed for the second order linear differential equation (DE) of the form

$$x''(t) = f(t)x(t) + u(t) \tag{1}$$

Due to a special structure of the DE, the fourth order error cancels and leads to sixth order algorithm using second order integration scheme. If we expand x(t) to some higher power and take into account the time reversal symmetry of the equation, all odd term cancel

$$x(h) = x(0) + hx'(0) + \frac{1}{2}h^2x''(0) + \frac{1}{3!}h^3x^{(3)}(0) + \frac{1}{4!}h^4x^{(4)}(0) + \frac{1}{5!}h^5x^{(5)}(0) + ...$$

$$x(-h) = x(0) - hx'(0) + \frac{1}{2}h^2x''(0) - \frac{1}{3!}h^3x^{(3)}(0) + \frac{1}{4!}h^4x^{(4)}(0) - \frac{1}{5!}h^5x^{(5)}(0) + ...$$

$$x(h) + x(-h) = 2x(0) + h^2(f(0)x(0) + u(0)) + \frac{2}{4!}h^4x^{(4)}(0) + O(h^6) \tag{4}$$

If we are happy with $O(h^4)$ algorithm, we can neglect $x^{(4)}$ term and get the following recursion relation

$$x_{i+1} - 2x_i + x_{i-1} = h^2(f_i x_i + u_i). \tag{5}$$

But we know from the differential equation that

$$x^{(4)} = \frac{d^2 x''(t)}{dt^2} = \frac{d^2}{dt^2}(f(t)x(t) + u(t)) \tag{6}$$

which can be approximated by

$$x^{(4)} \sim \frac{f_{i+1}x_{i+1} + u_{i+1} - 2f_i x_i - 2u_i + f_{i-1}x_{i-1} + u_{i-1}}{h^2} \tag{7}$$

Inserting the fourth order derivative in the equation (4), we get

$$x_{i+1}-2x_i+x_{i-1} = h^2(f_i x_i+u_i)+\frac{h^2}{12}(f_{i+1}x_{i+1}+u_{i+1}-2f_i x_i-2u_i+f_{i-1}x_{i-1}+u_{i-1}) \tag{8}$$

If we switch to a new variable $w_i = x_i(1 - \frac{h^2}{12}f_i) - \frac{h^2}{12}u_i$ we are left with the following equation

$$w_{i+1} - 2w_i + w_{i-1} = h^2(f_i x_i + u_i) + O(h^6) \tag{9}$$

The variable x needs to be recomputed at each step with $x_i = (w_i + \frac{h^2}{12}u_i)/(1 - \frac{h^2}{12}f_i)$.

The algorithm is surprisingly simple to implement as one needs only few lines of code. Here is the example for $u = 0$ (usual Schroedinger equation):

```
# Python implementation
def Numerov(f,x0,dx,dh):
    x = zeros(len(f))
    x[0] = x0
    x[1] = x0+dh*dx

    h2 = dh**2
    h12 = h2/12.

    w0 = x[0]*(1-h12*f[0])
    w1 = x[1]*(1-h12*f[1])
    xi = x[1]
    fi = f[1]
    for i in range(2,len(f)):
        w2 = 2*w1-w0+h2*fi*xi
        fi = f[i]
        xi = w2/(1-h12*fi)
        x[i]=xi
        w0 = w1
        w1 = w2
    return x
```

```
// C++ implementation
template <class funct>
void Numerov(funct& F, int Nmax, double x0, double dx, std::vector<double>& Solution)
{// Numerov algorithm for integrating the SODE of the form  x''(t)=F(t)x(t)
  // Solution[0] and Solution[1] need to be set (starting points)
  double h2 = dx*dx;  //square of step size
  double h12 = h2/12;  // defined for speed
  double w0 = (1-h12*F(x0))*Solution[0]; // first value of w
  double x = x0+dx;
  double Fx = F(x);
  double w1 = (1-h12*Fx)*Solution[1];    // second value of w
  double X = Solution[1];
  double w2;
  for (int i=2; i<Nmax; i++){
    w2 = 2*w1 - w0 + h2*X*Fx; // new value of w
    w0 = w1;
    w1 = w2;
    x += dx;
    Fx = F(x);                  // only one evaluation of F per step
    X = w2/(1-h12*Fx);        // new solution
    Solution[i] = X;
  }
}
```