

# Fortsetzung: 4. Gewöhnliche Differentialgleichungen

## Randwertproblem

**Erinnerung:** Typisch sind zwei Arten von Problemstellungen:

- **Anfangswertproblem:** der Funktionswert ist zu einem **bestimmten “Zeitpunkt”**  $t_0$  gegeben und die DGL beschreibt die Veränderung der Funktion ausgehend von diesem Zeitpunkt. ✓
- **Randwertproblem:** Funktionswerte sind teilweise zu verschiedenen “Zeiten” vorgegeben und eine Lösung soll diese **Randbedingungen** erfüllen.

Wir betrachten nun das **Randwertproblem**.

Wir nehmen an, dass die Differentialgleichung die Form einer  
**linearen homogenen/inhomogenen DGL 2. Ordnung**

hat

$$u''(x) + g(x) u(x) = s(x)$$

mit Randbedingungen  $u(x_1) = u_1$  und  $u(x_2) = u_2$

## Zwei typische Beispiele:

- Inhomogenes Problem: **Poisson-Gleichung** für das Potential  $\Phi(r) = \frac{\phi(r)}{r}$  bei der Ladungsverteilung  $\rho(r)$

$$\phi''(r) = -4\pi r \rho(r)$$

mit den Randbedingungen  $\phi(0) = 0$  und  $\lim_{r \rightarrow \infty} \frac{\phi(r)}{r} = 0$

- Homogenes Problem: **Schrödinger-Gleichung** für einen Bindungszustand der Energie  $E$  mit Bahndrehimpuls-Quantenzahl  $l$  im Potential  $V(r)$  und mit Masse  $m$

$$u''(r) + \left[ \frac{2m}{\hbar^2} (E - V(r)) - \frac{l(l+1)}{r^2} \right] u(r) = 0$$

mit den Randbedingungen  $u(0) = 0$  und  $\lim_{r \rightarrow \infty} u(r) = 0$

Die Beispiele haben die oben angenommene allgemeine Form

$$u''(x) + g(x) u(x) = s(x)$$

## Nebenbemerkung zur Form der DGL

Der Term mit einfacher Ableitung kann oft durch Umschreiben eliminiert werden.  
Angenommen

$$u''(x) + p(x) u'(x) + q(x) u(x) = s(x)$$

wobei die Stammfunktion  $P(x)$  von  $p(x)$  bekannt sei.

Mit dem Ansatz

$$u(x) = \exp\left(-\frac{1}{2}P(x)\right) z(x)$$

findet man durch einsetzen in die DGL, dass

$$z''(x) + g(x) z(x) = \tilde{s}(x)$$

Die Funktionen  $g(x)$  und  $\tilde{s}(x)$  lassen sich ebenfalls aus  $q(x), p(x)$  und  $P(x)$  bestimmen

$$\begin{aligned} g(x) &= q(x) - \frac{1}{2} p'(x) - \frac{1}{4} p^2(x) \\ \tilde{s}(x) &= \exp\left(+\frac{1}{2}P(x)\right) s(x) \end{aligned}$$

Diese Idee wurde bereits in den Beispielen genutzt um die einfachen Ableitungen zu entfernen.

Mit dem **Einfachschieß-** oder **Schießverfahren** kann man das **Randwertproblem** auf ein **Anfangswertproblem** zurückführen

Idee (für das inhomogene Problem):


- Formuliere das Problem als **Anfangswertproblem** bei  $x_1$

$$\begin{pmatrix} y^1(x)' \\ y^2(x)' \end{pmatrix} = \begin{pmatrix} y^2(x) \\ -g(x) y^1(x) \end{pmatrix}$$

mit Startwerten  $y^1(x_1) = u_1$  und  $y^2(x_1) = v_1$

- $u_1$  ist **Randbedingung** des ursprünglichen Problems  
 $v_1$  ist **freier Parameter**
- Löse das Anfangswertproblem für beliebiges  $v_1$  und bestimme
$$f(v_1) \equiv y^1(x_2) - u_2 = u(x_2) - u_2$$
- Finde mit einem Nullstellensuchverfahren eine Nullstelle von  $f(v_1)$

Beim homogenen Problem ist die Randbedingung nur für bestimmte Werte der Parameter erfüllbar (z.B. für bestimmte Energien  $E$  in der Schrödingergleichung)

Wie der zweite Parameter das Problem beim Schießverfahren beeinflusst ist dabei nicht wichtig. Außer den Startwerten von mehreren Komponenten (wie oben) kann man auch den Startwert und den Funktionswert **nach einem Schritt** als Parameter wählen  d.h. man kann ein sogenanntes **Mehrschrittverfahren** anwenden.

Ein Beispiel hierfür ist das **Numerov-Verfahren**. Die zweite Ableitung wird hierbei durch

$$\frac{u_{n+1} - 2u_n + u_{n-1}}{h^2} = u_n'' + \frac{h^2}{12} u_n'''' + O(h^4)$$

approximiert (durch Taylor-Entwicklung) und die Ableitung auf der rechten Seite durch

$$\begin{aligned} u''''(x) &= \frac{d^2}{dx^2} (-g(x) u(x) + s(x)) \\ &= - \frac{g_{n+1}u_{n+1} - 2g_n u_n + g_{n-1}u_{n-1}}{h^2} + \frac{s_{n+1} - 2s_n + s_{n-1}}{h^2} + O(h^2) \end{aligned}$$

Damit erhält man einen Zusammenhang der Funktionswerte  $u_{n+1}$ ,  $u_n$  und  $u_{n-1}$

$$\begin{aligned} &\left(1 + \frac{h^2}{12}g_{n+1}\right) u_{n+1} - 2 \left(1 - \frac{5h^2}{12}g_n\right) u_n + \left(1 + \frac{h^2}{12}g_{n-1}\right) u_{n-1} \\ &= \frac{h^2}{12} (s_{n+1} + 10 s_n + s_{n-1}) + O(h^6) \end{aligned}$$

- Das Numerov-Verfahren ist sehr viel **effizienter** als ein Runge-Kutta-Verfahren ähnlicher Ordnung
- die Linearität der Gleichung wurde ausgenutzt
- da Numerov ein Mehrschrittverfahren ist, eignet es sich besonders für das Schießverfahren

Beispiel der Poisson-Gleichung bei gaussförmiger Ladungsverteilung:

Potential  $\Phi(r) = \frac{\phi(r)}{r}$  erfüllt die DGL  $\phi''(r) = -4\pi r \rho(r)$

mit den Randbedingungen  $\phi(0) = 0$  und  $\lim_{r \rightarrow \infty} \frac{\phi(r)}{r} = 0$

Wir wählen die Ladungsverteilung  $\rho(r) = \frac{1}{r_0^3 \pi^{3/2}} \exp\left(-\left(\frac{r}{r_0}\right)^2\right)$

Wir wollen das Schießverfahren mit Numerov anwenden.

Eine Besonderheit an diesem Beispiel ist die Randbedingung

$$\lim_{r \rightarrow \infty} \frac{\phi(r)}{r} = 0$$

Hier ist es nützlich, sich eine allgemeine Lösung der inhomogenen Gleichung für **große r** anzuschauen (identisch zur homogenen):

$$\phi(r)'' = 0 \quad \longrightarrow \quad \phi(r) = Ar + B = B \quad (\mathbf{A=0} \text{ folgt aus der Randbedingung})$$

Das bedeutet, dass wir für große r annehmen können, dass die Lösung konstant ist. Die Konstante kann dann unser Parameter sein.

D.h. aber auch, dass wir die DGL von großen  $r$  zu  $r=0$  integrieren müssen. Die symmetrische Formulierung des Numerov Verfahrens lässt sich leicht entsprechend umformulieren.

$$u_{n-1} = \left(1 + \frac{h^2}{12}g_{n-1}\right)^{-1} \left[ \frac{h^2}{12} (s_{n+1} + 10 s_n + s_{n-1}) + 2 \left(1 - \frac{5h^2}{12}g_n\right) u_n - \left(1 + \frac{h^2}{12}g_{n+1}\right) u_{n+1} \right]$$

Durch Variation der Konstante  $B$  suchen wir die Lösung, für die die zweite Randbedingung  $\phi(0) = 0$  erfüllt ist.

```
/* beispiel-4.4.c Datum: 06.05.2020 */
```

```
/* dieser Code implementiert ein vereinfachtes Numerov Problem,  
   Erweiterung notwendig bei "Korrektur notwendig fuer g!=0 !!!!!" */
```

```
...
```

```
/* Code für secant, include statements etc. */
```

```
...
```

```
/* wegen Nullstellensuche definiere globale Felder, die in  
   Funktion zur Nullstellensuche genutzt werden koennen */
```

```
double r0bound;  
*/
```

```
/* definiert radius der homogenen Ladungsverteilung
```

```
double *r_array,*g_array,*s_array,*y_loesung; /* Zeiger auf Felder fuer Numerov und Loesung */  
double schrittweite_h; /* benutzte Schrittweite */  
int num_r; /* Anzahl der Stuetzstellen */
```

```
double gfunc(double r)
```

```
/* zu loesende Gleichung: y''(r)+g(r)*y(r)=s(r)  
   hier Definition der Funktion g(r) */
```

```
{  
    return 0.0;  
}
```

```
double sfunc(double r)
```

```
// zu loesende Gleichung: y''(r)+g(r)*y(r)=s(r)  
// hier Definition der Funktion s(r)
```

```
{  
  
    return -4.0*r*M_PI*exp(-(r/r0bound)*(r/r0bound))/  
           (r0bound*r0bound*r0bound*sqrt(M_PI*M_PI*M_PI));  
}
```

**kompletter Code nutzt Nullstellensuche mit secant (siehe Beispiel 3.3)**

**Globale Variablen werden in Funktion benötigt, deren Nullstelle mit secant gesucht werden soll.**

**definiere s(x) und g(x) für die Poisson-Gleichung mit**

$$\rho(r) = \frac{1}{r_0^3 \pi^{3/2}} \exp \left( - \left( \frac{r}{r_0} \right)^2 \right)$$



```
double init_numerov(double a,double b,int n,double *r,double *g,double *s)
```

**init\_numerov legt  $n$  Stützstellen  $r$  für das Intervall  $[a,b]$  fest und berechnet  $g$  und  $s$  und gibt Schrittweite zurück**

```
...
```

```
void numerovup(double *r,double *g,double *s,double h,int n,int steps,double
```

```
...
```

```
void numerovdown(double *r,double *g,double *s,double h,int n,int steps,double yn,double ynm1,double *y)
```

```
/* Funktion nutzt das Numerov Verfahren um fuer gegebene Stuetzstellen  $r$  und Funktionen  $g$  und  $s$  die Loesung  $y$  zu finden.
```

```
 $r$ ,  $g$  und  $s$  sollten mit init_numerov vorbereitet werden
```

```
 $h$  ist die Schrittweite (auch aus init_numerov)
```

```
 $n$  ist die Anzahl der Stuetzstellen
```

```
 $steps$  legt fest, wieviele Numerovschritte ausgefuehrt werden
```

```
 $yn$  und  $ynm1$  sind die Startwerte  $y[n-1]$  und  $y[n-2]$ , die Routine legt dann  $y[n-3] \dots y[n-steps-2]$  fest
```

```
 $y$  ist ein Feld der Laenge  $n$ , das die Loesungen bei Rueckkehr enthaelt
```

```
*/
```

```
{
```

```
int i;
```

```
double fakt_u_npl,fakt_u_n,fakt_u_nml; /* Variablen fuer Numerov Faktoren */
```

```
double fakt_s;
```

```
 $y[n-1]=yn$ ; /* belege erste Funktionswerte mit den Startwerten
```

```
 $y[n-2]=ynm1$ ;
```

**setze die Startwerte bei  $b$  und  $b - h$**

```
for(i=n-2; i>n-steps-2; i--) /* betrachte in den Schritten  $y(i-1)$  und  $y(i)$  um  $y(i+1)$  zu berechnen */
```

```
{
```

```
fakt_u_npl=1.0; /* Faktor bei  $y(i+1)$ ,
```

```
fakt_u_n=1.0; /* Faktor bei  $y(i)$ ,
```

```
fakt_u_nml=1.0; /* Faktor bei  $y(i-1)$ 
```

```
fakt_s=h*h/12.0*(s[i+1]+10.0*s[i]+s[i-1]);
```

**berechne für  $i=n-2, \dots, n-steps-2 (=0)$  die Funktion bei  $i-1$  und speichere in  $y$  (Achtung zur Zeit nur  $g=0$ !)**

```
 $y[i-1]=(fakt_s+2.0*fakt_u_n*y[i]-fakt_u_npl*y[i+1])/fakt_u_nml$ ; /* Rueckwaertsiteration */
```

```
}
```

```
}
```

**Funktion gibt Abweichung  
von der gewünschten Randbedingung (=0)  
bei  $r=0$  abhängig von para (= Konstante B)**

```
double f_randbed(double para)
/* Nullstelle dieser Funktion signalisiert
konsistent mit der Randbedingung ...
para ist der freie Parameter der Loesung
hier: para ist der Konstantewert der Loesung
Die Funktion nutzt globale Variablen, um die
r, g, s, h und n zu erhalten
und Feld y_loesung fuer die Loesung */
{
    numerovdown(r_array,g_array,s_array,schrittweite_h,num_r,num_r-2,para,para,y_loesung);
    return y_loesung[0]; /* zweite Randbedingung ist y[0] = 0 */
}
```

**nutze numerovdown um DGL mit Startwerten  
 $y[n]=y[n-1]=B$  zu lösen.**

**gebe  $y[0]$  zurück = Nullstelle von  $y[0]$**

```
int main()
{ double rmax,para,zero; /* max. r, Randbedingung und Null aus Auswertung der Randbed. funktion */
  int i,num_schritt; /* Anzahl der Sekantenverfahren */
  double rho; /* fuer Ausgabe der Ladungsverteilung */
  ...
  schrittweite_h=init_numerov(0.0,rmax,num_r,r_array,g_array,s_array); /* belege r,g und s mit Werten */

  /* suche mit dem Schiessverfahren und mit N ...

  para=secant(-1.0, 1.0, &f_randbed,&num_schritt);

  zero=f_randbed(para);

  /* Ausgabe der Loesung phi/r und Vergleich mit para, ...
  printf("# %15s %15s %15s \n","r","Phi","para/r");
  for(i=1; i<num_r; i++)
  {
      printf(" %15.6e %15.6e %15.6e \n",r_array[i],y_loesung[i]/r_array[i],para/r_array[i]);
  }
  printf("#Parameter,Null,Schritte %15.6e %15.6e %d\n",para,zero,num_schritt);
  ...
}
```

**Deklarationen und Speicherplatz mit malloc**

**definiere r,g,s und h mit init\_numerov**

**finde korrekte Randbedingung mit secant  
und (wichtig!) bestimme Lösung für  
korrekten Parameter para.**

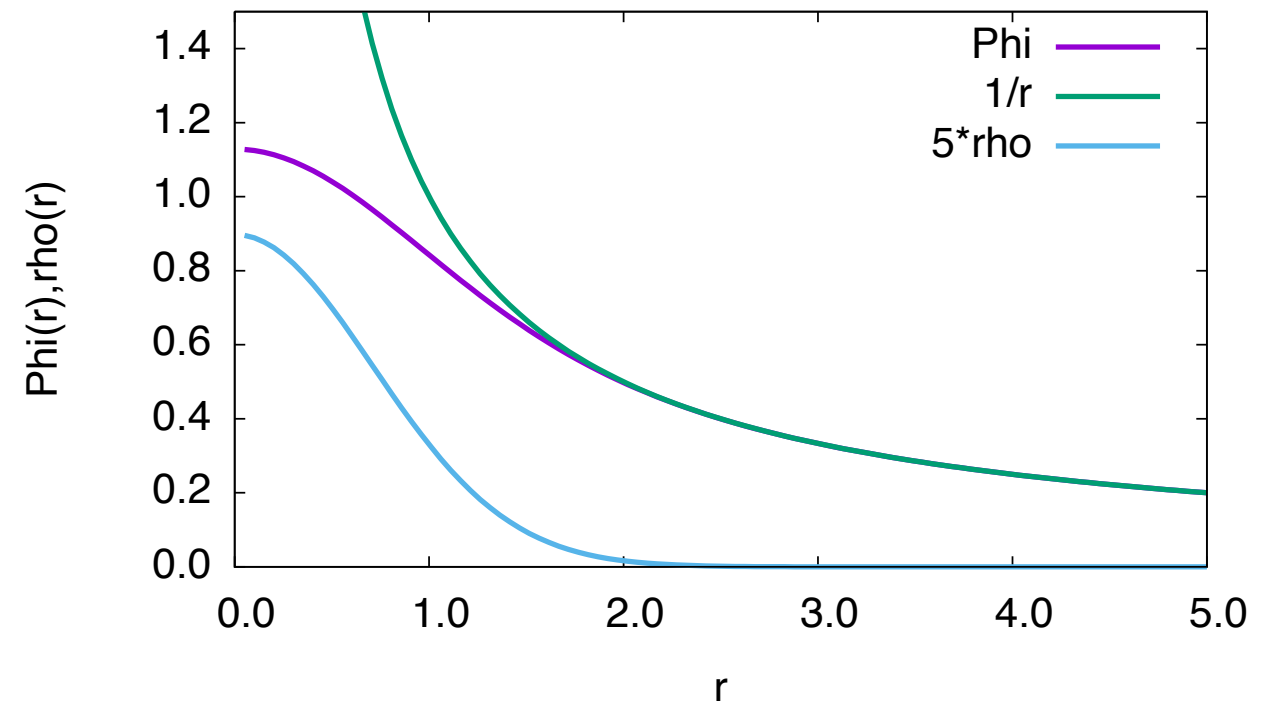
**gebe Lösung und etwas Statistik aus**

# Geben Sie  $r_{\max}, r_0$  und die Anzahl der Stuetzstellen ein: 5 1 100

#	$r$	$\Phi$	$\text{para}/r$
	5.050505e-02	1.127421e+00	1.980000e+01
	1.010101e-01	1.124554e+00	9.900000e+00
	1.515152e-01	1.119804e+00	6.600000e+00
...			
	4.898990e+00	2.041237e-01	2.041237e-01
	4.949495e+00	2.020408e-01	2.020408e-01
	5.000000e+00	2.000000e-01	2.000000e-01

#Parameter, Null und Anzahl der secant Schritte: 1.000000e+00 -1.387779e-17 2

- $1/r$  Verhalten bei großen  $r$
- Abhängigkeit von  $B$  linear, secant braucht nur zwei Schritte
- Inhomogenität ist Null bei großen  $r$
- Konstante  $B$  entspricht Gesamtladung mit
- hoher Genauigkeit!
- Implementierung der Randbedingung hängt beim Schießverfahren vom physikalischen Problem ab



Numerov (und alle Problem auf Basis einer Taylor-Entwicklung) funktioniert nicht gut bei unstetigen Problemen, z.B. konstanter Ladungsverteilung für  $r < r_0$ .

## Inhomogenes Randwertproblem mit Green's Funktion

Wir betrachten immer noch die linearen inhomogenen DGL 2. Ordnung

$$u''(x) + g(x) u(x) = s(x)$$

mit Randbedingungen  $u(x_1) = 0$  und  $u(x_2) = 0$

Wir nehmen an, dass wir zwei homogenen Lösungen  $v(x)$  und  $w(x)$  bereits kennen. Die beiden Lösungen erfüllen die Randbedingungen

$$v(x_1) = w(x_2) = 0$$

Mit der **Wronski Determinante** (konstant, da  $\frac{dW[v, w](x)}{dx} = 0$ )

$$W_0 \equiv W[v, w](x) \equiv v(x)w'(x) - v'(x)w(x)$$

definiert man die **Green'sche Funktion**

$$G(x, y) = \frac{1}{W_0} \begin{cases} v(x)w(y) & \text{für } x \leq y \\ v(y)w(x) & \text{für } x > y \end{cases}$$

mit der man die Lösung mit den korrekten Randbedingung durch

$$u(x) \equiv \int_{x_1}^{x_2} dy G(x, y) s(y) = \frac{1}{W_0} \left[ w(x) \int_{x_1}^x dy v(y) s(y) + v(x) \int_x^{x_2} dy w(y) s(y) \right]$$

erhält.

Das kann man durch Ableitung überprüfen

$$\begin{aligned}
 u'(x) &= \frac{1}{W_0} \left[ w'(x) \int_{x_1}^x dy v(y)s(y) + \cancel{w(x)v(x)s(x)} + v'(x) \int_x^{x_2} dy w(y)s(y) - \cancel{v(x)w(x)s(x)} \right] \\
 u''(x) &= \frac{1}{W_0} \left[ \color{blue}{w''(x)} \int_{x_1}^x dy v(y)s(y) + \color{red}{w'(x)v(x)s(x)} + \color{blue}{v''(x)} \int_x^{x_2} dy w(y)s(y) - \color{red}{v'(x)w(x)s(x)} \right] \\
 &= \frac{1}{W_0} \left[ \color{red}{W_0 s(x)} - \color{blue}{g(x)w(x)} \int_{x_1}^x dy v(y)s(y) - \color{blue}{g(x)v(x)} \int_x^{x_2} dy w(y)s(y) \right] \\
 &= s(x) - g(x) u(x)
 \end{aligned}$$

Wegen der Integralgrenzen und Randbedingungen der homogenen Lösungen gilt auch

$$u(x_1) = 0 \quad \text{und} \quad u(x_2) = 0$$

Damit findet man die Lösung durch einfache Integration.

## Beispiel

$$u''(x) + g(x) u(x) = s(x) \quad \text{mit} \quad s(x) = -\frac{1}{2} x \exp(-x) \quad \text{und} \quad g(x) = -\omega^2$$

Randbedingungen:  $u(0) = 0$  und  $\lim_{r \rightarrow \infty} u(r) = 0$

Erster Schritt: identifiziere Lösungen der homogenen Gleichung  
(hier analytisch möglich, auch numerische Lösung kann sinnvoll sein)

$$v(x) = \sinh(\omega x) \quad \text{und} \quad w(x) = \exp(-\omega x)$$

Analytische Lösung des Problems ist bekannt

(numerische Lösung mit Schießverfahren in Übungen)

➔  $u(x) = \frac{1}{8} x(1+x) \exp(-x) \quad \text{bzw.} \quad u(x) = e^{-\omega x} - \frac{\left(1 + \frac{1}{2} (-\omega^2 + 1) x\right) e^{-x}}{(-\omega^2 + 1)^2}$

Code erfordert Implementierung des Ausdrucks (numerische Integration)

$$u(x) \equiv \int_{x_1}^{x_2} dy G(x, y) s(y) = \frac{1}{W_0} \left[ w(x) \int_{x_1}^x dy v(y) s(y) + v(x) \int_x^{x_2} dy w(y) s(y) \right]$$

```
/* Datei: beispiel-4.5.c Author: BCM / C version AN Datum: 08.05.2012 */
```

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<float.h>
#include<limits.h>
#include<string.h>
#include<ctype.h>
```

**Code implementiert Green's Funktions Methode**

```
/* Vorwaertsdeklaration einer Funktion
```

**Benutzt Routine zur Auswertung von  
Kommandozeilen Parameter: GetUserParam(...)  
Vorwärtsdeklaration (z.B. in Header-Datei)**

```
/* Eingabe ueber die Kommandozeile auswerten: Routine wird gleich beschrieben
   Ziel ist Eingabeparameter der Kommandozeile in globalen Variablen zu speichern */
```

```
void GetUserParam( int, char *[] );
```

```
/* Globale Variablen: */
```

**globale Variablen werden in GetUserParam  
verändert (je nach Kommandozeile)**

```
long    n=100;      /* # Schritten fuer I
double omega=1.0;   /* Parameter der Funktion g(x) */
double b=10;        /* Obere Grenze fuer die Integration */
```

```
/* Definition der DGL durch Inhomogenit
```

**Definiert DGL und Randbedingungen:  
homogene Lösungen und Inhomogenität**

```
double sf ( double r ) {
    /* Inhomogenitaet der Differentialg
    return -0.5 * r * exp(-r);
}
double v ( double r ) {
    /* bei r=0 regulaere Loesung der homogenen Gleichung */
    return sinh( omega * r );
}
double w ( double r ) {
    /* bei lim_r->\infty regulaere Loesung der homogenen Gleichung */
    return exp( -omega * r );
}
double sol (double r) /* analytische Loesung
...
double Wronski () { /* Wronski-Determinante v
    return -omega;
}
```

**Code nutzt analytisch bekannte Lösung  
zur Ausgabe**

**Wronski Determinante ist konstant**



```

int main( int argc, char *argv[] ) {
/* argc = # Argumente, argv enthaelt die Argumente, argv[0] enthaelt Programmname */
/* Deklaration der Variablen */
...
  GetUserParam( argc, argv ); /* Parameter
...
  h = b / (double) n; /* Schrittweite */
  c = 0.5 * h / Wronski();
...
  int1 = 0; r = 0;
  sm1 = sf(0); vm1 = v(0);

  phi[0] = 0;
  for (i=1; i<n; i++) {
    r += h;
    v0 = v(r);
    s0 = sf(r);
    int1 += c * ( sm1 * vm1 + s0 * v0 );
    phi[i] = int1 * w(r);
    sm1 = s0;
    vm1 = v0;
  }

  int2 = 0; r = n * h;
  sp1 = sf(r); wp1 = w(r);

  phi[n] = 0;
  for (i=n-1; i>0; i--) {
    r -= h;
    w0 = w(r);
    s0 = sf(r);
    int2 += c * ( wp1 * sp1 + s0 * w0 );
    phi[i] += int2 * v(r);
    sp1 = s0;
    wp1 = w0;
  }
...

  return 0;
}

```

main hat Parameter argc und argv!  
Argumente von GetUserParam

Schrittweite und Gewicht innerhalb eines  
Intervalls

phi[i] nutzt Integral von phi[i-1]

$$u_1(x) = \frac{1}{W_0} w(x) \int_{x_1}^x dy v(y) s(y)$$

phi[i] nutzt Integral von phi[i+1]

$$u_2(x) = \frac{1}{W_0} v(x) \int_x^{x_2} dy w(y) s(y)$$



argc: Anzahl der Parameter

argv: Vektor von Zeigern auf Strings mit argc Elementen

```

void GetUserParam( int argc, char *argv[] )
    int i;          /* Variablen: */
    char* endptr;
    const char usage[] = "# beispiel-4.5 [-n <Schrittzahl>] [-w <Frequenz>] [-b <Obere Grenze>]";
    const char error_message[] = "# FEHLER(GetuserParam): falsche Option: ";

    if (argc>1) /* falls es ueberhaupt
    {
        for (i=1; i<argc; i++)
        {
            /* parameter 2 Charakter lang und sollte mit '-' anfaengen ... */
            if ( (strlen(argv[i])==2) && (argv[i][0] == '-') )
            {
                switch (argv[i][1])
                {
                    case 'w':
                        omega = strtod( argv[++i], &endptr);
                        if ( (!isspace(*endptr) && (*endptr) != 0) ) {
                            printf(" %s \n %s \n",error_message,usage); exit(1); }
                        break;
                    case 'b':
                        b = strtod( argv[++i], &endptr);
                        if ( (!isspace(*endptr) && (*endptr) != 0) ) {
                            printf(" %s \n %s \n",error_message,usage); exit(1); }
                        break;
                    case 'n':
                        n = strtol( argv[++i], &endptr, 10);
                        if ( (!isspace(*endptr) && (*endptr) != 0) ) {
                            printf(" %s \n %s \n",error_message,usage); exit(1); }
                        break;
                    default:
                        printf(" %s \n %s \n",error_message,usage);
                        exit(1);
                }
            } else
            {
                printf(" %s \n %s \n",error_message,usage);
                exit(1);
            } /* end of: if-then-else */
        } /* end-of: for */
    } /* end-of: if */
}

```

gehe durch alle Argumente

1. Teil hat Form "-x" 2. Teil ist meist Zahl (bei uns)

für jedes mögliche Argument (-w -b -n)  
wir der nächste Parameter ausgewertet

switch wählt einen von mehreren Fällen

strtod (für double) und strtol (für long)

strto... gibt Wert zurück und in  
endptr einen Zeiger auf nicht mehr  
ausgewertete Zeichen (sollte 0 sein)Werte werden hier in globalen Variablen  
gespeichert

Durch die Routine `GetUserParam` ist es möglich im Terminal Parameter der numerischen Lösung zu testen:

```
> gcc -lm beispiel-4.5.c -o beispiel-4.5
> ./beispiel-4.5 -w 1.0 -n 100 -b 10.0
# green: # ===== # omega =      1.00000e+00, n =      100, b =      1.00000e+01,
h =      1.00000e-01
#
#           r                      phi(r)
#
0.0000000000e+00      0.0000000000e+00      0.0000000000e+00
1.0000000000e-01      1.2479190403e-02      1.2441514498e-02
2.0000000000e-01      2.4630103621e-02      2.4561922592e-02
3.0000000000e-01      3.6207427207e-02      3.6114888258e-02
. . .
```

Für schnelle Prüfung der Konvergenz ist auch ein direkter Aufruf innerhalb gnuplots oder andere Plot-Programmen möglich, z.B.

```
> gnuplot
```

. . .

```
gnuplot> plot "<./beispiel-4.5 -w 1.0 -n 10 -b 10.0" with lines, "<./beispiel-4.5 -w 1.0  
-n 20 -b 10.0" with lines, "<./beispiel-4.5 -w 1.0 -n 30 -b 10.0" with lines
```

Man kann so schnell die Konvergenzeigenschaften erkennen.

Hier sieht man, dass die Green'sche Funktion mit etwa 30 Stützstellen eine akzeptable Lösung der DGL liefert.

