Cornell University

Department of Physics

Phys 480/680, Astro 690 February 8, 2000

# Computational Physics, Spring 2000

## Homework Assignment # 2

(Due Thursday, February 17 at 8:40 am in class.)

## Agenda and readings for the week of February 1:

The readings are from *Numerical Recipes: The Art of Scientific Computing*, 2nd edition (in Fortran *or* C).

- Lec4, 02/08: Theory of errors in numerical integration; digit shifting tests; Richardson extrapolation. **Reading: 4.2–4.3, Intro to 4.4**
- Lec5, 02/10: Theory of solution of ODE's; discussion of application to Poisson's equation **Reading: 16.0–16.1**
- Lec6, 02/15: Root finding via bisection; Schrodinger's equation for atoms. **Reading: 9.0-9.2, 17.0-17.2**
- Lab2, 02/17: Convergence of solution of ODE's; numerical solutions of Schrödinger's equation; stability.

## Contents


**1 Extending "rk4" and "derivs" 2**
- 1.1 Coding . . . 2
- 1.2 Debugging . . . 2

**2 Solving Poisson's equation 3**
- 2.1 Kernel for Poisson's equation . . . 3
- 2.2 Poisson's equation solver . . . 3
- 2.3 Solve for the potential of the 1s state of hydrogen . . . 3

**3 HW#2 ADDENDUM 4**
- 3.1 Double precision . . . 4
  - 3.1.1 Integration routines . . . 4
  - 3.1.2 "rk4ind" . . . 4
  - 3.1.3 Poisson equation solver . . . 5
- 3.2 "User friendly" code . . . 5
  - 3.2.1 Scaling with $Nu$ . . . 5
  - 3.2.2 Poisson's equation for the 1s state of hydrogen . . . 5
- 3.3 Schrödinger's equation . . . 5


**Note:** For the programs in this problem set we face a small notational difficulty. We have been using the variable $n(r)$ to represent the number of electrons per unit volume throughout space. It would make sense then to define an array $n[]$ to hold the values of the electron density. However, $n$ is so commonly used as an indexing variable (particularly in *Numerical Recipes*) that this is likely to cause difficulties. Thus, we shall use the array $rho[]$ to represent the electron density $n(r)$.

# 1 Extending "rk4" and "derivs"

## 1.1 Coding

Copy (carefully) the numerical recipes routine "rk4", rename it "rk4ind", and extend it to allow derivs to access the arrays *rho*, *r*, *dr*:

- Add five new arguments to the prototype for "derivs" to produce

```
C:       void derivs(float x, float y[], float dydx[],
                     int k, float rho[], float r[], float dr[], int Nu)

Fortran: subroutine derivs(x,y,dydx,k,rho,r,dr,Nu)
         implicit none
         real x,y(*),dydx(*),rho(Nu+1),r(Nu+1),dr(Nu+1)
         integer k,Nu
```

- Add six new arguments to the "rk4ind" prototype and update its prototype for derivs to produce

```
C:       void rk4ind(float y[], float dydx[], int n, float x, float h, float yout[],
               void (*derivs)(float, float [], float [], int, float[], float[], float[], int),
               int k, int dk, float rho[], float r[], float dr[],int Nu)

Fortran: subroutine rk4ind(y,dydx,n,x,h,yout,derivs,k,dk,rho,r,dr,Nu)
         implicit none
         real y(n),dydx(n),x,h,yout(n),rho(Nu+1),r(Nu+1),dr(Nu+1)
         integer n,k,dk,Nu
         external derivs
```

- Within "rk4ind", treat the indices $k$ and $dk$ as integer versions of $x$ and $h$: (1) when rk4ind defines $xh = x + hh$ with $hh = h/2$, define $kh = k + dk/2$, (2) whenever rk4ind calls derivs add the five new arguments to the call, (3) make sure that the appropriate value for k (either $k$, $kh$ or $k + dk$) is passed to each call to derivs, paralleling how rk4ind treats $x$.

## 1.2 Debugging

As we discussed in class, using fourth-order Runge-Kutta to solve $F'(x) = g(x)$ is equivalent to integrating $\int g(x)\,dx$ with Simpson's rule. Debug your version of "rk4ind" by comparing the output of the following code fragment with the output of your "simpint" routine for an *appropriately* designed "derivs" routine:

```
              .
              .
              .
[Define f, r, dr, and Nu]
              .
              .
              .
   h=2./float(Nu)
   dk=2
   v(1)=0.
   do k=1,Nu-1,2
      x=(k-1)*h
      call dphi(x,v,dv,k,f,r,dr,Nu)
      call rk4ind(v,dv,1,x,h,v,dphi,k,dk,f,r,dr,Nu)
   enddo
```

```
write(6,*) simpint(f,r,dr,Nu),v(1)
          .
          .
          .
```

# 2 Solving Poisson's equation

## 2.1 Kernel for Poisson's equation

Design an appropriate "derivs" routine to implement the first-order version of Poisson's equation as discussed in class,

$$\frac{d}{du}\begin{pmatrix} \Phi \\ \Phi' \end{pmatrix} = \begin{pmatrix} \Phi' \\ -4\pi n(r) r \end{pmatrix} \cdot \frac{dr}{du},$$

where $\phi(r) = \Phi(r)/r$.

## 2.2 Poisson's equation solver

Using your subroutine (2.1) and a code fragment similar to that given for (1.2), write a main program which solves Poisson's equation.

**Notes:** Be sure to integrate backwards from $r = \infty$ ($u$=1) to $r = 0$ ($u$=0): you'll have to change the order of the loop and the values of *both* $h$ and $dk$! Also, remember the boundary conditions:
$\Phi(\infty) = \Phi(r(u = 1)) = Q$ and $\Phi'(\infty) = \Phi'(r(u = 1)) = 0$.

## 2.3 Solve for the potential of the 1s state of hydrogen

Problem 2 of Homework #1 gave a formula for the potential for the $1s$ state of hydrogen. Recall that the electron density for this state was

$$n_{1s}(\vec{x}) = \psi_{1s}^2(\vec{x}) = \frac{1}{\pi} e^{-2r}.$$

Use your solver to compute the potential $\phi_{1s}(r)$ which arrises from this density and compare your results with the formula given in Homework #1!!!

# 3 HW#2 ADDENDUM

## 3.1 Double precision

From our experience in Lab #1, it best to develop numerical software in double precision. Accordingly, we should update all of our codes to work in double precision.

### 3.1.1 Integration routines

Update your integration routines "trapint" and "simpint" from HW#1 so that they now function in double precision:

```
C:        double trapint(double f[],double r[],double dr[],int N)

Fortran:  double precision function trapint(f,r,dr,N)
          integer N
          double precision f(N),r(N),dr(N)
```

and

```
C:        double simpint(double f[],double r[],double dr[],int N)

Fortran:  double precision function simpint(f,r,dr,N)
          integer N
          double precision f(N),r(N),dr(N)
```

**Notes:** This should involve mostly replacing all occurrences of "float" with "double" with an editor (if your code is in C) or replacing "real" with "double" (if your code is in Fortran). Also, if you are coding in Fortran, you must make sure that your numerical constants, like "2." or "3.", are in double precision. The best way to do this is to put "d0" after them, as in "2.d0" and "3.d0".

### 3.1.2 "rk4ind"

Update your ODE routines in Problems 1.1 to function in double precision:

```
C:       void derivs(double x, double y[], double dydx[],
                     int k, double rho[], double r[], double dr[], int Nu)

Fortran: subroutine derivs(x,y,dydx,k,rho,r,dr,Nu)
         implicit none
         double precision x,y(*),dydx(*),rho(Nu+1),r(Nu+1),dr(Nu+1)
         integer k,Nu
```

and

```
C:       void rk4ind(double y[], double dydx[], int n, double x, double h, double yout[],
               void (*derivs)(double, double [], double [], int, double[], double[], double[], int),
               int k, int dk, double rho[], double r[], double dr[],int Nu)

Fortran: subroutine rk4ind(y,dydx,n,x,h,yout,derivs,k,dk,rho,r,dr,Nu)
         implicit none
         double precision y(n),dydx(n),x,h,yout(n),rho(Nu+1),r(Nu+1),dr(Nu+1)
         integer n,k,dk,Nu
         external derivs
```

**Notes:** Again, this should involve mostly replacing all occurrences of "float" with "double" or replacing "real" with "double" and, if your code is in Fortran, making sure all numerical constants are in double precision. Also, if you are programming in C, be sure to replace the calls to the "vector" utility with calls to "dvector".

### 3.1.3 Poisson equation solver

Be sure that your code for the Poisson solver (Problem 2.2 of HW#2) is also working in double precision!

## 3.2 "User friendly" code

To minimize the time we actually have to spend coding in lab, make sure that your code has the following capabilities:

### 3.2.1 Scaling with $Nu$

Make sure that for your debugging code in Problem 1.2 (of HW#2)

- both simpint and rk4ind work in double precision;
- your code is set up to do the integral $V_{el-nuc} = \int_0^\infty (-1/r)(e^{-2r}/\pi)(4\pi r^2)\, dr$;
- your code accepts $Nu$ as input from the *keyboard* (so that you don't have to keep recompiling), for values up to 100,000; and
- your code outputs its results to at least 15 digits (C: "%20.15f"; Fortran: "f20.15").

### 3.2.2 Poisson's equation for the 1s state of hydrogen

For Problem 2.3 (of HW#2), make a plot which shows both your solution ($\phi(r(u))$ vs. $r(u)$) and the analytic formula

$$\phi_{1s}(r) = \frac{e^{-2r} - 1}{r} + e^{-2r}.$$

Also, so that we can investigate convergence, make sure your code

- takes $Nu$ as input from the *keyboard* for values up to 100,000; and
- prints out the value of $\Phi(r)$ ($\equiv r\phi(r)$) as computed at the origin ($r = 0$) to at least 15 digits (C: "%20.15f"; Fortran: "f20.15").

## 3.3 Schrödinger's equation

Write an alternate code which takes the parameter $E$ as input from the *keyboard* and solves the equation

$$\frac{d}{du}\begin{pmatrix} \Psi \\ \Psi' \end{pmatrix} = \begin{pmatrix} \Psi' \\ 2(-1/r - E)\Psi \end{pmatrix} \cdot \frac{dr}{du},$$

with boundary conditions $\Psi(u = 1) = 0$, $\Psi'(u = 0) = 1e - 20$, for $Nu = 200$.

This code should integrate the equations inward from infinity just as does your Poisson solver, and should be a relatively straightforward generalization of your code for Problem 2.2.