Scientific Computing (Phys 2109/Ast 3100H)
I. Scientfic Software Development

SciNet HPC Consortium

University of Toronto

November 2011

SciNet
compute • calcul
CANADA

Part III

Structures in C, Simple ODE solvers

SciNet
compute • calcul
CANADA

Homework 2 discussion

HW2 - Common non-pgplot pitfalls

What goes in a header file again, and why?

- Function declarations and constants (and a few others we will see today).
- So including allows the code to use those functions and constants.
- Header guards to prevent double inclusion:

```
#ifndef _MODULEH_
#define _MODULEH_

/* code */

#endif
```

SciNet compute • calcul CANADA

HW2 - Common non-pgplot pitfalls

Unit testing

- Not the same as full program (integrated) testing.
- Focus on one function and write a test for it.
- Floating point and precision: Floating point is not of unlimited precision.
- Comparing two floating points using $==$ is a bad idea. Put in a tolerance $\mathbf{O(10^{-5})}$.

SciNet
compute • calcul
CANADA

HW2 - Common non-pgplot pitfalls

tar

- Tar or zip your files up before you send them in
- Easier for us
- Less error-prone for you.

Example

```
$ tar zcvf hw3.tgz *.c *.h Makefile README
```

SciNet compute • calcul CANADA

HW2 - Building Pgplot

Some pointers

After makemake, edit Makefile:

- replace g77 with gfortran
- change FFLAGC with reasonable values
- may need to remove the line "pndriv.o:"
- Make sure devel. packages for X and png are installed.
- When linking, you need `-lcpgplot -lpgplot -lgfortran -lX11 -lpng`. The order matters, and these libraries should be the last arguments.

SciNet compute • calcul CANADA

Linking to libraries

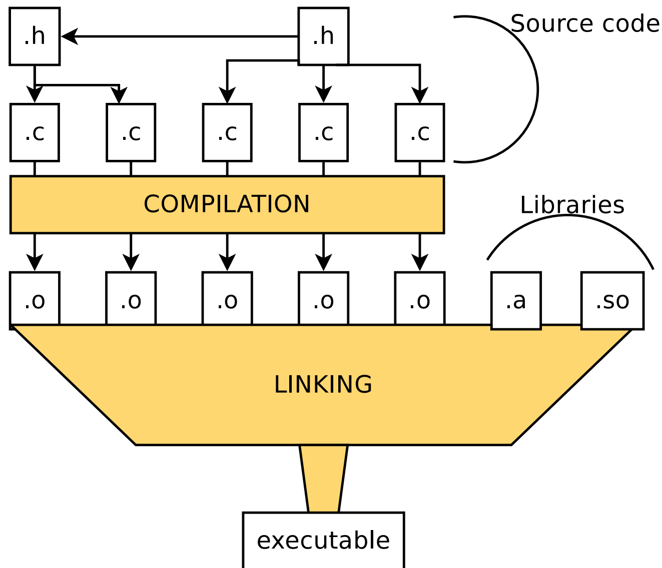
You will likely be linking to external libraries for a variety of reason:

- graphics
- faster/optimized math libraries
- blas
- fft

How does one do this in general?

SciNet
compute • calcul
CANADA

Linking to libraries



SciNet compute • calcul CANADA

Linking to libraries

At compilation stage

For your code to use the library, it needs to include the corresponding header file.

```
#include "libraryname.h"
```

```
$ gcc -O3 -Ilibraryincludepath myfile.c -o myfile.o
```

At linking stage

The actual library is like an object file and has to be linked in. There are in fact two types:

- Static(.a): are included in the executable (like .o files)
- Dynamic(.so): not in executable, loaded at startup

```
$ gcc myfile.o -Llibrarypath -llibraryname
```

SciNet
compute • calcul
CANADA

Linking to libraries

Example: link diffuse2 with pgplot

- First compile with

```
-DPGPLOT
```

  including a -I argument if needed.
- Link with the following additional libraries:

```
-lcpgplot -lpgplot -lgfortran -lX11 -lpng
```

  Add a -L argument before it if needed.

SciNet compute • calcul CANADA

Linking to libraries

Example: link diffuse2 with pgplot

- First compile with

```
-DPGPLOT
```

  including a -I argument if needed.

- Link with the following additional libraries:

```
-lcpgplot -lpgplot -lgfortran -lX11 -lpng
```

  Add a -L argument before it if needed.

⇒ HANDS-ON

SciNet
compute • calcul
CANADA

Where are we going with this?

1. 2D diffusion for density field $\rho(\mathbf{r}, \mathbf{t})$ governed by PDE

$$\frac{\partial \rho}{\partial t} = D\left(\frac{\partial^2 \rho}{\partial x^2} + \frac{\partial^2 \rho}{\partial y^2}\right). \tag{1}$$

2. Tracer particle satisfies ODE

$$m\ddot{\mathbf{R}} = \mathbf{F} - \alpha(\rho)\dot{\mathbf{R}}, \tag{2a}$$

   where $\mathbf{m}$ is the mass, $\mathbf{F}$ is a force acting on the particle and the friction constant α is (proportional to) the viscosity.

   - Ad hoc form for density dependent friction constant α:

$$\alpha(\rho) = \alpha_0(1 + a\rho). \tag{2b}$$

   - Ad hoc form for force, like a constant electric field:

$$\mathbf{F} = qE\hat{\mathbf{x}}. \tag{2c}$$

3. Periodic boundary conditions in all directions, i.e.,

$$\mathbf{r} \sim \mathbf{r} + L(n\hat{\mathbf{x}} + m\hat{\mathbf{y}}). \tag{3}$$

   where $\mathbf{L}$ is the length of the side of the periodic box.

SciNet
compute • calcul
CANADA

Where are we going with this?

1. 2D diffusion for density field $\rho(\mathbf{r}, \mathbf{t})$ governed by PDE

Module

$$\frac{\partial \rho}{\partial t} = D \left(\frac{\partial^2 \rho}{\partial x^2} + \frac{\partial^2 \rho}{\partial y^2} \right). \quad (1)$$

2. Tracer particle satisfies ODE

$$m\ddot{\mathbf{R}} = \mathbf{F} - \alpha(\rho)\dot{\mathbf{R}}, \quad (2a)$$

where $\mathbf{m}$ is the mass, $\mathbf{F}$ is a force acting on the particle and the friction constant α is (proportional to) the viscosity.

- Ad hoc form for density dependent friction constant α:

$$\alpha(\rho) = \alpha_0(1 + a\rho). \quad (2b)$$

- Ad hoc form for force, like a constant electric field:

$$\mathbf{F} = qE\hat{x}. \quad (2c)$$

3. Periodic boundary conditions in all directions, i.e.,

$$\mathbf{r} \sim \mathbf{r} + L(n\hat{x} + m\hat{y}). \quad (3)$$

where $\mathbf{L}$ is the length of the side of the periodic box.

SciNet compute • calcul CANADA

Where are we going with this?

1. 2D diffusion for density field $\rho(\mathbf{r}, t)$ governed by PDE

Module

$$\frac{\partial \rho}{\partial t} = D\left(\frac{\partial^2 \rho}{\partial x^2} + \frac{\partial^2 \rho}{\partial y^2}\right). \quad (1)$$

2. Tracer particle satisfies ODE

$$m\ddot{R} = F - \alpha(\rho)\dot{R}, \quad (2a)$$

where m is the mass, F is a force acting on the particle and the friction constant α is (proportional to) the viscosity.

Module

- Ad hoc form for density dependent friction constant α:

$$\alpha(\rho) = \alpha_0(1 + a\rho). \quad (2b)$$

- Ad hoc form for force, like a constant electric field:

$$F = qE\hat{x}. \quad (2c)$$

3. Periodic boundary conditions in all directions, i.e.,

$$r \sim r + L(n\hat{x} + m\hat{y}). \quad (3)$$

where L is the length of the side of the periodic box.

SciNet compute • calcul CANADA

Modularity and language constructs in C

- Modules will contain lot of variables for that module only.
- Do not want to use global (or static) variables.
- How to group variables then?

in C, you can use structs.

SciNet
compute • calcul CANADA

C Variables

Define a variable with

```
type name;
```

where *type* may be a

- built-in type:
  - floating point type:
    float, **double**, **long double**
  - integer type:
    short, [**unsigned**] **int**, [**unsigned**] **long int**
  - character or string of characters:
    char, **char***
- array
- pointer
- structure

SciNet compute • calcul CANADA

C structs

Structures: collection of other variables.

```
struct name {
   type1 name1;
   type2 name2;
   ...
};
```

SciNet compute • calcul CANADA

C structs

Structures: collection of other variables.

```
struct name {
   type1 name1;
   type2 name2;
   ...
};
```

Example

```
struct Info {
   char name[100];
   unsigned int age;
};
struct Info myinfo;
myinfo.age = 38;
strcpy(myinfo.name, "Ramses");
```

Struct definitions go in the header file!

SciNet compute • calcul CANADA

C structs

Typedefs

Used to give a name to an existing data type, or a compound data type.

```
typedef existingtype newtype;
```

Similar to `existingtype name;` but defines a type instead of a variable.

SciNet
compute • calcul
CANADA

C structs

Typedefs

Used to give a name to an existing data type, or a compound data type.

```
typedef existingtype newtype;
```

Similar to `existingtype name;` but defines a type instead of a variable.

Example (a way to get rid of the `struct` keyword)

```
typedef struct Info Info_t;
```

Then you can declare a `struct Info` simply by

```
Info_t myinfo;
```

SciNet compute • calcul CANADA

C structs

Typedefs

Used to give a name to an existing data type, or a compound data type.

```
typedef existingtype newtype;
```

Similar to `existingtype name;` but defines a type instead of a variable.

Example (a way to get rid of the `struct` keyword)

```
typedef struct Info Info_t;
```

Then you can declare a `struct Info` simply by

```
Info_t myinfo;
```

Type definitions go in the header file!

SciNet compute • calcul CANADA

C structs

Pointers to structs

Imagine the trouble of calling an element *element* of a struct when given a pointer *ptr* to that strcut:

```
(*ptr).element
```

This is confusing and prone to typos.
There an easier syntax for this:

```
ptr->element
```

This is particularly useful in functions.

SciNet
compute • calcul
CANADA

C structs - example 1

```
void theoryCalc(float time,float **rho,float *x,int
npnts,float a0,float sigma0,float d,float x1,float
x2,int nimages);
```

SciNet
compute • calcul
CANADA

C structs - example 1

```
void theoryCalc(float time,float **rho,float *x,int
npnts,float a0,float sigma0,float d,float x1,float
x2,int nimages);
```

```
typedef struct {
   float **rho;
   int npnts;
} Rho;
typedef struct {
   float *x;
   int npnts;
   float x1;
   float x2;
} Grid;
typedef struct {
   int nimages;
   float a0;
   float sigma0;
} Theory;
void theoryCalc(float time,Rho*rho,Grid*x,Theory*start);
```

SciNet
compute • calcul
CANADA

C structs - example 2

```
float theoryError(float **rho1,float **rho2,int npnts){
   float error = 0;
   for (int i = 1; i <= npnts; i++)
      for (int j = 1; j <= npnts; j++)
         error += pow(rho1[i][j] - rho2[i][j],2);
   return sqrt(error);
}
```

SciNet
compute • calcul
CANADA

C structs - example 2

```
float theoryError(float **rho1,float **rho2,int npnts){
   float error = 0;
   for (int i = 1; i <= npnts; i++)
      for (int j = 1; j <= npnts; j++)
         error += pow(rho1[i][j] - rho2[i][j],2);
   return sqrt(error);
}
```

```
float theoryError(Rho* a, Rho* b){
   float error = 0;
   for (int i = 1; i <= a->npnts; i++)
      for (int j = 1; j <= a->npnts; j++)
         error += pow(a->rho[i][j] - b->rho[i][j],2);
   return sqrt(error);
}
```

SciNet compute • calcul CANADA

Ordinary differential equations

General form

$$\frac{dx}{dt} = f(x, t)$$

ODEs pop-up in lots of places

- Trajectories of molecules, celestial bodies
- 1d stationary soluations of PDEs
- Population dynamics
- ...

Here, we'll look only at ODEs with initial conditions:
i.e. $\mathbf{x(t = 0)}$ given.

Still leaves an enormous class of system.

SciNet compute • calcul CANADA

Ordinary differential equations

Basic algorithm

- Discretize time curve $\mathbf{x(t)}$
- Link the discrete elements (time points)
- Evaluate $\mathbf{f}$ along the way

Usually time stepping:

$$\mathbf{t} \longrightarrow \mathbf{t}' = \mathbf{t} + \delta\mathbf{t}$$
$$\mathbf{x} \longrightarrow \mathbf{x}'$$

such that

$$\mathbf{x}' - \mathbf{x}(\delta\mathbf{t}) = \mathcal{O}(\delta\mathbf{t}^{\mathbf{k}+1})$$

$\mathbf{k}$ is the order.

SciNet
compute • calcul
CANADA

Ordinary differential equations

Example (Forward Euler)

```
EULER ALGORITHM
SET x to the initial value x(0)
SET t to the initial time
WHILE t < tfinal
   COMPUTE f(x,t)
   UPDATE x to x+f(x,t)*dt
   UPDATE t to t+dt
END WHILE
```

Usually not very good and easily becomes unstable.
Order 1.

SciNet compute • calcul CANADA

Ordinary differential equations

- There exist general algorithms to solve ordinary differential equations numerically (see e.g. *Numerical Recipes* Ch. 16), such as Runge-Kutta and predictor/correction algorithms.
- Many of these are too costly or not stable enough for long simulations of many-particle systems.
- In MD simulations, it is therefore better to use algorithms specifically suited for systems obeying Newton's equations of motion, such as the Verlet (or leap flog) algorithm.
- In other situations, RK may do.
- One then usually takes variable time steps.

SciNet
compute • calcul
CANADA

Ordinary differential equations

Particle dynamics usually a second order differential equation:

$$\ddot{x} = f(x, t) \tag{4}$$

To handle this:

- define $\dot{x}$ as a variable, so this becomes a set of coupled odes
- use a scheme for second order odes

Example (Verlet algorithm, 3rd order in position)

$$x_{n+1} = 2x_n - x_{n-1} + f_n \frac{\delta t^2}{m}$$

```
SET time t to 0
WHILE t < tfinal
   COMPUTE the force f
   COMPUTE new position xnew=2*x-xprev+f*dt*dt/m
   UPDATE previous position xprev to x
   UPDATE position x to xnew
   UPDATE t to t+dt
END WHILE
```

velocities: $\dot{x} \approx (x_n - x_{n-1})/\delta t$ (crude).

SciNet compute • calcul CANADA

Handson 2

Write program for tracer particle in 2d with fixed friction coefficient

$$\mathbf{m\ddot{R} = qE\hat{x} - \alpha_0 \dot{R}},$$

where $\mathbf{m}$ is the mass, $\mathbf{qE}$ is an electric force acting on the particle and the friction constant α_0 is proportional to the viscosity.

- Periodic boundary conditions in all directions, such that coordinates restricted to lie between $\mathbf{0}$ and $\mathbf{L}$.
- Initial conditions: $\mathbf{R(0) = R_0}$ and $\mathbf{\dot{R}(0) = V_0}$.
- Parameter values:
  $\mathbf{D = 1; m = 1; \alpha_0 = 1; qE = 1; L = 10; R_0 = 0; V_0 = 10\hat{y}}$

Use structs and the Verlet scheme.

SciNet compute • calcul CANADA

Homework assignment #3

- Finish the interpolation and the test
- Rewrite program for tracer particle to work with a given density field such that $\alpha_0 \rightarrow \alpha(\rho(\mathbf{x})) = \alpha_0(1 + \mathbf{a}\rho(\mathbf{x}))$, with a $=$15.
- Link the two so the density evolves at the same time as the tracer particle.

Will send a more detailed assignment later today...

SciNet
compute • calcul
CANADA