(1)

Lecture 3

~~Arrays, 1d and 2d~~

Arrays - matrices

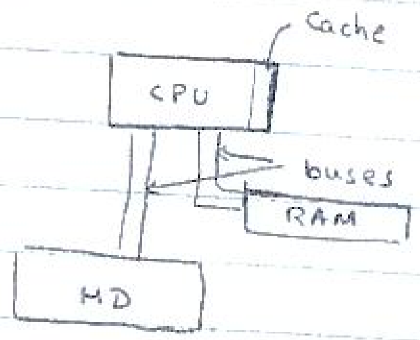
<u>Recap</u>: Fortran stores its arrays in column major order. In other words

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{pmatrix}$$

$$(a_{11} \; a_{21} \cdots a_{m1} \; a_{12} \; a_{22} \cdots a_{m2} \cdots \cdots a_{1n} \cdots a_{mn})$$

Why should it matter?

It should matter because of how a computer actually computes.



A very rough sketch.

Ex Consider the following piece of code:

```
program add

  real 4 :: a, b, c
  ! reserves place in RAM for three number (32 bit
float) a, b, c, but not fill them up
     a = 1.
     b = 5.
  ! write these two numbers to the two
    memory locations that were reserved.
     c = a + b
  ! copy 'a' to cache
  ! copy 'b' to cache
  ! add a + b and write that in cache
  ! copy c from cache to memory location
  ! that was reserved.
end program
```

The memory in RAM are organised like a book, in pages. When the computer loads a number from RAM to cache it loads the whole page.
So numbers at adjacent memory locations are immediately available in the cache.

~~So~~ # ~~p~~

CPU can compute much faster than it can access RAM so it pays if you do many computation for one RAM access. This means to optimize cache access.

In fortran a multidimensional array must always be accessed such that its first index is the fastest varying one.

```
do f(64, 64, 64, 3)

do nvec = 1, 3
   do k = 1, nz
      do j = 1, ny
         do i = 1, nx
            f(i, j, k, nvec) =
         enddo
      enddo
   enddo
enddo
```

we shall test whether that is actually useful or not by finding the dot product of two vector arrays.

```
do nvec; do k;  do j;  do i;
                  ~~c(i;~~
                  ~~c(:,j,k~~
   enddo  enddo enddo enddo
```

Replace by:

```
do k;  do j;
       c(:,j,k) = a(:,j,k,1) * b(:,j,k,1)
                + a(:,j,k,2) * b(:,j,k,2)
                + a(:,j,k,3) * b(:,j,k,3)
       enddo
enddo
```

Whether this actually helps in a practical situation is always to be tested.

To test we need to profile our code.

The simplest way is to use the time command.

we shall deal with more ~~complx~~ elaborate methods as we go along

Finding the largest eigenvalue and the corresponding eigenvector of a real-symmetric matrix.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

Construct the quadratic form

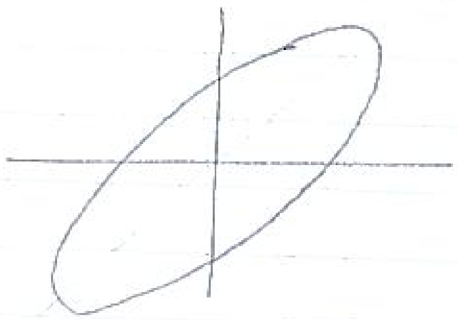
$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$= \begin{pmatrix} a_{11} x_1 + a_{21} x_2 & a_{12} x_1 + a_{22} x_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$= a_{11} x_1^2 + a_{21} x_1 x_2 + a_{11} x_1 x_2 + a_{21} x_2^2$$

$+$ other terms

$=$ Eqn of an ellipse



Let us choose a vector and multiply it by this matrix. This vector

$$|y_0\rangle = b_1 |\lambda_1\rangle + b_2 |\lambda_2\rangle$$

$$A |y_0\rangle = b_1 \ell_1 |\lambda_1\rangle + b_2 \ell_2 |\lambda_2\rangle$$

$$= \ell_1 \left[b |\lambda_1\rangle + \left(\frac{\ell_2}{\ell_1}\right) b_2 |\lambda_2\rangle \right]$$

$$A^2 |y_0\rangle = \ell_1^2 \left[b_1 |\lambda_1\rangle + \left(\frac{\ell_2}{\ell_1}\right)^2 b_2 |\lambda_2\rangle \right]$$

$$\vdots$$

$$|y_n\rangle = A^n |y_0\rangle = \ell_1^n \left[b_1 |\lambda_1\rangle + \left(\frac{\ell_2}{\ell_1}\right)^n b_2 |\lambda_2\rangle \right]$$

$$\simeq \ell_1^n \; b_1 |\lambda_1\rangle$$

- By looking at the vector at large time and normalizing it we shall get $|\lambda_1\rangle$
- Once we know $|\lambda_1\rangle$ we can find ℓ_1 by multiplying the $|\lambda_1\rangle$ By A again

finding out by what factor the vector scales.

<u>Problems</u>

- $|\gamma_n\rangle$ can become very large
  sol^n : normalize at every step

- How to monitor convergence ?
  $|\gamma_{n+1}\rangle - |\gamma_n\rangle$? (in some norm)
  $\langle \gamma_{n+1} | \gamma_n \rangle$? (each normalized to unity)

- Accelarating convergence by shifting
  - $(A - s\mathbb{I})|\gamma\rangle = \cancel{\lambda}\,(\ell - s)|\gamma\rangle$

- Use different initial guess

- Accelarating convergence by Aitken's extrapolation
  $\{\gamma_1, \gamma_2, \ldots \gamma_n\}$ $\quad y_\infty = h$

$$(\gamma_\infty - \gamma_{n+1}) = c(\gamma_\infty - \gamma_n)$$ linear convergence

$$(y_\infty - y_{n+2}) = c\,(y_\infty - y_{n+1})$$

$$\frac{y_\infty - y_{n+1}}{y_\infty - y_n} + \frac{y_\infty - y_{n+2}}{y_\infty - y_{n+1}} = 0$$

$$(y_\infty - y_{n+2}) = -\frac{(y_\infty - y_{n+1})^2}{(y_\infty - y_n)}$$

After two iteration we can find y_∞ and shift there.

This prescription can be applied to each component of $|y\rangle$

Steps

(i) first write a simple code
(ii) ~~allow~~ break it up into subroutines.
(iii) add shifting
(iv) add accelaration of convergence.
(v) add different initial condition
(vi) Read in the matrix from an input file.