## Improving performance

Matlab's a pig

## Outline

- Announcements:
  - Homework I: Solutions on web
  - Homework II: on web--due Wed.
- Homework I
- Performance Issues

## Homework I

- Grades & comments are waiting in your mailboxes
  - PASS--you passed!
    - try to learn from your mistakes
  - PROVISIONAL--you passed, but I'm watching
    - 2 or more provisional passes will make it difficult for me to let you pass
  - FAIL--you're failing and need to see me ASAP!

## Homework I

- Most everyone did well on 1-4
  - check the comments I sent
- 5+ somewhat harder

## Homework

- "Essential knowledge" questions should be fairly easy
  - just the basics covered in lecture
- "Programming" questions will be harder
  - apply what we've talked about to a real problem
- The goal of the problem sets is to build your skill and confidence
  - I don't intend for this to be painful
  - If you find that you're spending several hours on a problem, please see me.

## Fourier Series--Problems 5-7

- We can represent a function x(t) by sines & cosines
- Define a vector of times t for which we want to know the value of x.
  - t and x are vectors of the same size.
- The jth entry in x will be its value at time t(j) is given by:

$$x(j) = \sum_{k=1}^{N/2+1} a_k \cos(\frac{2\pi(k-1)t(j)}{Ndt}) + b_k \sin(\frac{2\pi(k-1)t(j)}{Ndt})$$

## Fourier Series--Problems 5-6

- To keep things simple, let's ignore sine terms and pretend the cosines don't exist:

$$x(j) = \sum_{k=1}^{N/2+1} a_k \frac{2\pi(k-1)t(j)}{Ndt}$$

- Can implement this as a double loop:

```
n=length(a); N=2*n-2; p=length(t);
x=zeros(p,1); f=2*pi/(N*dt);
for j=1:p
  for k=1:n
    x(j)=x(j) + a(k)*f*(k-1)*t(j);
  end
end
```

## Fourier Series--Problems 5-6

- Inner loop looks a lot like a vector product c=a'*b:

```
c=0;
for k=1:n
   c=c + a(k)*b(k);
end
```

- Can eliminate inner loop:

```
n=length(a); N=2*n-2; p=length(t);
x=zeros(p,1); f=2*pi/(N*dt);
K=[1:n]-1;
for j=1:p
     x(j)=f*t(j)*K*a(:);
end
```

## Fourier Series--Problems 5-6

- If we can use vector * to eliminate one loop, why not the other?
  - Multiplying t & K gives a p-by-n matrix in which each row is K scaled by an element of t:

```
t(:)*K  = [t(1)*K;
           t(2)*K;
              :
           t(p)*K ]
```

  - If we multiply this matrix by a, we get the desired form for x

```
t(k)*K*a(:) = [t(1)*K*a);
               t(2)*K*a;
                  :
               t(p)*K*a ]
```

## Problem Set II

- You must implement the scheme we developed (in key) as a function
  - Inputs: a, b, t
  - Outputs: x
- You will create another function that will solve for a and b
  - Inputs: x, t
  - Outputs: a, b, f

## Performance

- Factors affecting performance:
  - Overhead--time to find a function, check it, and start it
    - Error checking, polymorphism adds to overhead
  - Memory--time to allocate memory to variables
  - FLOPS--how much math do you do?

## Overhead

- Matlab has inherently high overhead compared to compiled languages (C, FORTRAN)
  - Matlab checks each command in an m-file one-by-one
    - only once/session unless code is changed
  - C-compiler checks commands once during compilation
- Matlab spends time locating functions
- Matlab creates a workspace for each function

## Minimizing Overhead

- Can translate into a compiled language
  - Usually straightforward
  - Matlab compiler will generate C code (not necessarily what you would write, though)
- Use subfunctions:

```
file fname.m:
    function O=fname(I)
        :
    function O2=fname2(I2)
        :
```

  - fname2 is only available inside fname
  - Matlab checks fname2 with fname, spends less time trying to find it

## Minimizing Overhead

- Can avoid memory overhead by inlining functions
  - rather than calling function (or subfunction) fname2, replace calls with code for fname2 (make sure variable names are ok)
- This may increase performance, but it is BAD STYLE
  - Makes code harder to read, maintain, reuse

## Minimizing Overhead

- Use vectorized functions
  - for j=1:100;
    - sin(f(j));% must start-up sine function each time
  - end
  - sin(f); % much faster, especially for big f.
- In general, Matlab's built-in functions are faster
  - MathWorks employees are paid to write code
  - You are not!

## Memory

- Matlab arrays are allocated dynamically and can grow:
  - a=1; %a is 1-by-1
  - for j=2:1000;
    - a(j)=j; %a grows 1 double each time
  - end
- Much faster to allocate arrays before loop
  - a=ones(1,1000);
  - for j=2:1000;a(j)=j;end

## Flops

- It takes time to do math
  - * + - are fast, / and ^ are slower
- Try to "pre-compute" when possible
  - for j=1:100
    - x(j)=2*pi/3 * f(j) %2*pi/3 is computed each time
  - end

  - twopi3= 2*pi/3;
  - for j=1:100;
    - x(j)*twopi3*f(j);
  - end

## Fourier-like Example

- 4 functions
  - TwoLoop.m--two loops
  - OneLoop.m--replace inner loop with inner product
  - NoLoop.m--replaces outer loop with outer product
  - VecExp.m--performs timing experiment with these functions

## Some comments on performance

- The Three "E's"
  - Effective--does it solve the problem?
  - Efficient--how quickly?
  - Elegant--is it simple, easy to understand?
- Efficiency (speed) is only one goal.
- Time spent tuning code should be factored into performance
  - Spending 2 hours improving runtime from 10 min to 5 min only makes sense if you will use the code a lot or on much larger problems

## Survey

- You now know the basics of Matlab
  - The rest of the course will be spent extending and reinforcing that knowledge
- More Matlab or more applications?

| | | |
|---|---|---|
| Matlab | Polymorphic functions | |
| | Objects beyond arrays | |
| | Improving performance | |
| Applications | File I/O (binary & text) | |
| | Linear Systems | |
| | Diff. Equations | |
| | Statistics | |
| | Graphics | |
| | Polynomials & splines | |
| | Signal processing (FFT) | |
| | Optimization | |