DN2222
Applied Numerical Methods
- part 2:
Numerical Linear Algebra

Aim of the Course

Basic understanding of important details in applied numerical linear algebra.

- Linear systems of equations, $Ax = b$ (where A is $n \times n$).
- Linear least squares problems,
  $Ax = b$ (where A is $m \times n$, $m > n$).
- Eigenvalue problems.
  (Given A find x and λ such that $Ax = \lambda x$)
- Singular Value Decomposition

3 programming paradigms:

- Library routines (BLAS, EISPACK, LAPACK, NAG)
- Programming environment (Matlab, Mathematica)
- Templates

Already seen (parts) of this in the earlier courses.

Why do it (again)?

Reason is two-fold:

- Give an understanding of the functioning of the software. What can be expected from the code?
- To be able to select a good software for your application.

Simple example: Fibonacci numbers

- 1st Fibonacci number is 0
- 2nd Fibonacci number is 1
- All other Fibonacci numbers are the sum of the two immediately previous numbers

```
function f=fibo(n);
   if n==1;
      f=0;
   elseif n==2;
      f=1;
   else
      f=fibo(n-1)+fibo(n-2);
   end;
```

```
function f=fibos(n);
   v=zeros(1,n);
   v(2)=1;
   for i=3:n;
      v(i)=v(i-1)+v(i-2);
   end;
   f=v(n);
```

Solving $Ax = b$

- **Matrix factorization** - the matrix A provides the key information for the solution process.
- **Perturbation Theory and Condition numbers** - often given data has limited accuracy - then what to expect from the result?
- **Round off errors** - An algorithm is always performed on a computer with finite arithmetics (floating point numbers). How does this effect the result?

Mathematically equivalent

$$f(x) = \sqrt{(x + 1/x)} - \sqrt{x} - 1/(\sqrt{x})^3$$

$$g(x) = \frac{1/x}{\sqrt{(x + 1/x)} + \sqrt{x}} - 1/(\sqrt{x})^3$$

---

x	$f(x)$	$g(x)$
$1.0000e+01$	$-1.5851e-02$	$-1.5851e-02$
$1.0000e+02$	$-5.0001e-04$	$-5.0001e-04$
$1.0000e+03$	$-1.5811e-05$	$-1.5811e-05$
$1.0000e+04$	$-5.0000e-07$	$-5.0000e-07$
$1.0000e+05$	$-1.5811e-08$	$-1.5811e-08$
$1.0000e+06$	$-5.0001e-10$	$-5.0000e-10$
$1.0000e+07$	$-1.5707e-11$	$-1.5811e-11$
$1.0000e+08$	$-1.0000e-12$	$-5.0000e-13$
$1.0000e+09$	$-3.1623e-14$	$-1.5811e-14$

Solving $Ax = b$ (cont.)

- **Speed** - How long time does the code need to run? How much work and what kind of work is needed?
- **Engineering Numerical Software**
  - Ease of use
  - Reliability
  - Robustness
  - Speed

Simple integral

$$\int_a^b 123\mathrm{e}^{-(4567(1-\cos(x)))^2)}dx$$

```
i = quad('fkn',a,b)
```

a	b	i
$-\pi$	π	$4.6662e+00$
$-\pi+0.1$	$\pi+0.1$	$4.8015e-225$
$-\pi+0.1$	π	$4.0447e-13$
π	$\pi+0.1$	$0.0000e+00$

Organizational Details

- 8 lectures, 7 scheduled lab sessions
- Course Webpage:
  `http://www.csc.kth.se/utbildning/...`
  `...kth/kurser/DN2222/nummet2-11`
- Course Lecturer: Ninni Carlsund
- Course Assistant: Ashraful Kadir

Course Literature

- JW Demmel: Applied Numerical Linear Algebra SIAM 1997.
- A Ruhe: Topics in Numerical Linear Algebra KTH-CSC 2007.
- Some handouts

Course Requirements

- 3 lab assignments.
- 1 written examination (3 hours), Friday December 16th, 10-13 o'clock.

Plan

```
F1 Introduction,
   Foundations of Error Analysis.
      (L1,D1.2-3)
F2 Error Analysis (cont.), Gaussian Elimination
      (D2.1-4)
F3 Direct Methods for Sparse Matrices
      (L2, D2.7)
F4 Linear Least Squares Method,
   Singular Value Decomposition
      (L3, D3.1,2,5)
F5 Singular Value Decomposition (cont),
   Eigenvalues
      (L4, D4)
F6 Eigenvalues (cont)
      (L4, D4)
F7 Large Scale Problems:  Iterative Methods
      (L5, D6.6)
F8 Wrap-up
```

Sources of error

Answers from a numerical calculation are seldom exactly correct. There are two sources of error.

- There may be errors in the input data. The input often comes from measurements or previous calculations.
- There are errors caused by the algorithm itself, due to approximations within the algorithm.

In order to estimate how these errors effect the calculated answer, we need to understand how the solution changes when the input is slightly changed/perturbed.

Perturbation theory and Condition Number

- Let $f(x)$ be a real valued differentiable function.
- We want to compute $f(x)$ but we do not have x exactly. We are given z and δx knowing that $z-\delta x \leq x \leq z+\delta x$.
- The best we can do, without more information, is to compute $f(z)$ and try to bound the error $|f(x)-f(z)|$.
- Using linear approximation we get $|f(z)-f(x)| = |f(x+\delta x)-f(x)| \sim |\delta x| \cdot |f'(x)|$.
- We call $|f'(x)|$ the *absolute condition number*
- To get an relative estimation we rewrite the expression:

$$\frac{|f(x+\delta x)-f(x)|}{|f(x)|} \sim \frac{|\delta x|}{|x|} \cdot \frac{|f'(x)| \cdot |x|}{|f(x)|}$$

- The multiplier of $|\delta x|/|x|$ is called the *(relative) condition number.*

$$\kappa = \frac{|f'(x)| \cdot |x|}{|f(x)|}$$

Condition Number

- The condition number is all we need to tell us how errors in input data will affect the computed answer.
- A problem is *ill-conditioned* if the condition number is large.
- A problem is *ill-posed* if the condition number is infinite.
- A problem is *well-posed* if the condition number is finite.

Roundoff error analysis

- When computing $f(x)$ we get $\hat{y}$ rather than mathematical y.
- Mathematically $\hat{y}$ should be the result of $f(\hat{x})$.
- $|y - \hat{y}|$ is the forward error.
- $|x - \hat{x}|$ is the backward error.
- An algorithm is said to be backward stable if the backward error is small for all x ($alg(x) = f(x + \delta x)$).
- Informally: a backward stable algorithm gives the exact answer for a slightly wrong problem: $alg(x) = f(x + \delta x)$.
- Showing backward stability requires knowledge of roundoff error during basic floating point operations of the machine.

Analysing Speed

- We want to estimate the time a particular algorithm will need before we implement it.
- Traditionally one counted *flops*.
- We will compare flops for all our algorithms.
- However today this might be misleading since modern computers run parallel, but also need time to move data inside the computer.

Floating Point Arithmetics - Scientific Notation

- In *Scientific Notation* a number is described by sign, fraction, base and exponent: $-.31416 \times 10^1$.
- A floating point number is called normalized if the leading digit of the fraction is non-zero.
- Computers usually use the base 2.
- In normalized binary floating point numbers the leading digit is always 1 and need not be stored explicitly, giving space to store one more bit.

Storage

- Computers have historically had many different choices of base and length.
- Now IEEE standard for binary numbers is most common.
- IEEE standard for single precision is 32 bits (1 sign, 8 exponent, 23 fraction).
- IEEE standard for double precision is 64 bits (1 sign, 11 exponent, 52 fraction).

IEEE Single Precision

- IEEE standard for single precision is 32 bits (1 sign, 8 exponent, 23 fraction).
- The number is stored as $(-1)^s \cdot 2^{e-127} \cdot (1+f)$
- The maximum relative representation error is $2^{-24} \approx 6 \cdot 10^{-8}$
- The underflow threshold is $2^{-126} \approx 1 \cdot 10^{-38}$
- The overflow threshold is $2^{128} \approx 3 \cdot 10^{38}$

IEEE Double Precision

- IEEE standard for double precision is 64 bits (1 sign, 11 exponent, 52 fraction).
- The number is stored as $(-1)^s \cdot 2^{e-1023} \cdot (1+f)$
- The maximum relative representation error is $2^{-53} \approx 1 \cdot 10^{-16}$
- The underflow threshold is $2^{-1022} \approx 2 \cdot 10^{-308}$
- The overflow threshold is $2^{1024} \approx 2 \cdot 10^{308}$

Some review questions:

- **Q2.** When is a floating point number normalized?
- **Q4.** What is the overflow threshhold? What is its approximate value in IEEE double precision?
- **Q9.** What are well-posed and ill-posed problems?
- **Q10.** What are well-conditioned problems?
- **Q11.** Why is well-conditioning necessary for obtaining reliable results?
- **Q7.** How do you bound the rounding error when computing a sum of three numbers, $s = a+b+c$?
- **Q12.** What are forward and backward stability?

Next time (tomorrow):

- **Norms**
- **Gaussian Elimination**
- **Error Propagation**
- **Pivoting**

$/\mathcal{NC}$