CSC431/331 - Scientific Computing

class slides

Massimo Di Pierro

mdipierro@cs.depaul.edu

DePaul CTI

Scientific Computing

Definitions

- "Study of Algorithms for the problems of *continuous mathematics*" (as distinguished from *dicerete mathematics*) (from Wikipedia).
- "The study of approximation techniques for solving mathematical problems, taking into account the extent of possible errors" (from Answers.com)

Applications: physics, biology, finance, etc.

Scientific Computing Techniques

- Linear Algebra
- Solvers of non-linear equations
- Optimization / Minimization / Maximization
- Fitting
- Numerical Integration and Differentiation
- Differential Equations
- Fourier / Laplace transform
- Stochastic methods (csc521)

Well posed and stable problems

We deal with *well posed* problems:

- Solution must exist and be unique
- Sulution must depend continuously on input data

Most physical problems are well posed except at “critical points” where any infinitesimal variation in one of the parameters of the system may cause very different behavior of the system (chaos).

Well posed and stable problems

Algorithms work best with *stable* problems:

- Solution is weakly sensitive to input data (condition number < 1)

Condition Number

The condition number is a measure of sensitivity.

- input: x
- output: y
- problem: $y = f(x)$

$$\text{[condition number]} = \frac{\text{[relative } y\text{-error]}}{\text{[relative } x\text{-error]}} = |xf'(x)/f(x)|$$

- a problem is well-conditioned if $cn < 1$
- a problem is ill-conditioned if $cn > 1$

Strategies for Scientific Commputing

- Approximate continuum with discerete
- Replace integrals with sums
- Replace derivatives with finite differences
- Replace non-linear with linear + corrections
- Replace complicated with simple
- Transform a problem into a different one
- Approach the true result by iterations

Strategies for Scientific Commputing

Example of iterative algorithm:

```
result=guess
loop:
    compute correction
    result=result+correction
    if |reminder| < target_precision return result
    # often reminder < correction
```

Types of Errors

- Data error
- Computational error
- Systematic error (hard to control)
- Modeling error
- Rounding error
- Statistical error (can be controlled)
- Trucation error

[total error] $=$ [comput. error] $+$ [propagated data error]

Measuring Statistical error

$[\text{abslute error}] = [\text{approximate value}] - [\text{true value}]$

```
result=guess
loop:
    compute correction
    result=result+correction
    if |reminder| < target_absolute_precision return result
    # often reminder < correction
```

$[\text{relative error}] = [\text{absolut error}] / [\text{true value}]$

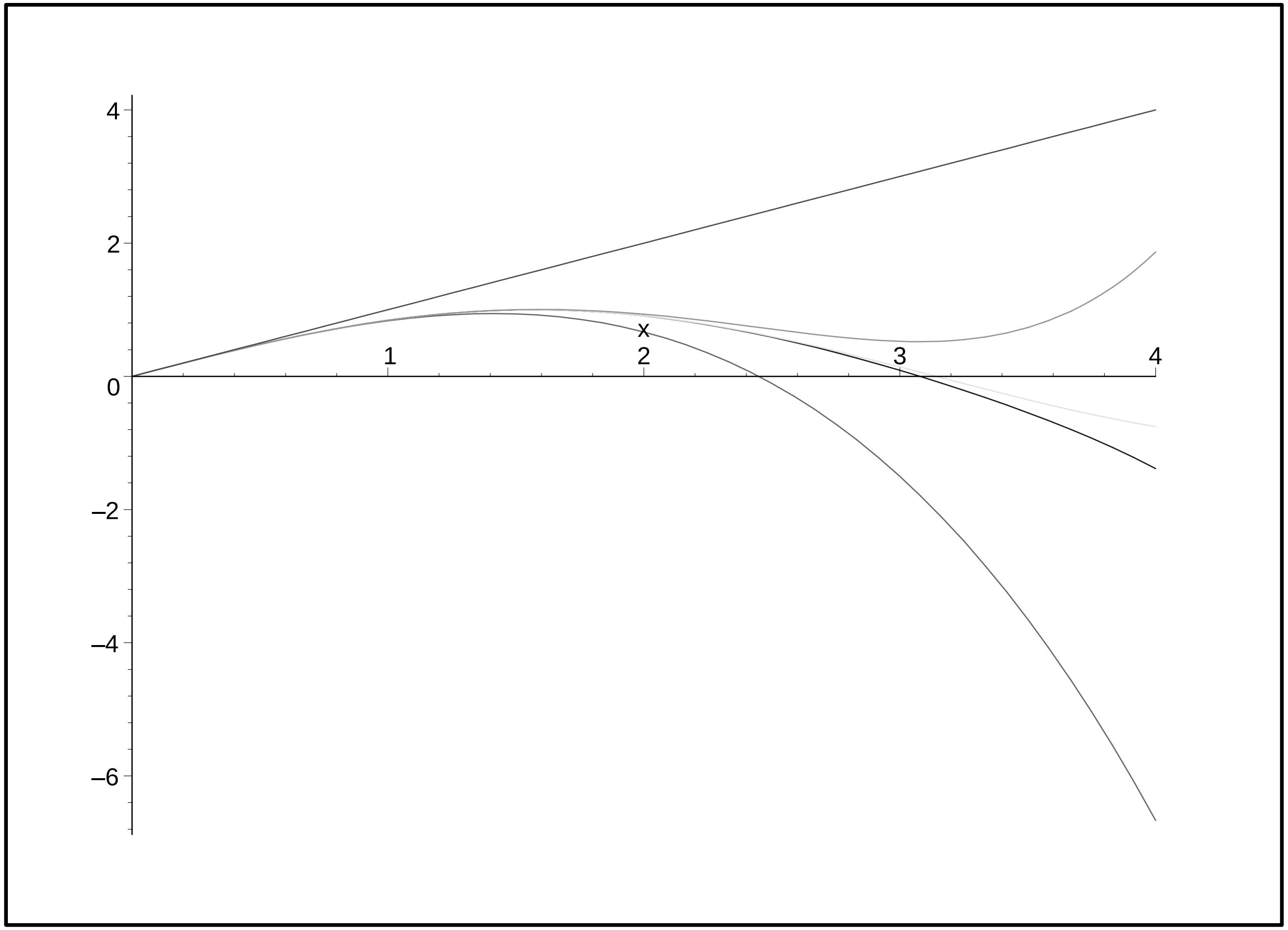
```
result=guess
loop:
    compute correction
    result=result+correction
    if |reminder/result| < target_relative_precision return result
    # often reminder < correction
```

Useful Maple commands

```
> 10+5;
> 100!;
> evalf(sin(1),10);
> solve( {x+y=2, x-y=5}, {x,y} );
> g:=diff(sin(x),x);
> int(g(x),x=0..Pi);
> sum( 3^i/i!, i=0..infinity);
> series(sin(x), x=0, 10);
> h:=convert(%,polynom);
> plot( {sin(x), h(x)}, x=-4..4);
> plot3d( sin(x+y)*x, x=-2..2, y=-2..2);
> with(LinearAlgebra);
> A:=Matrix(2,2,[[1,2],[3,4]]);
> b:=Matrix(2,1,[[4],[5]]);
> MatrixInverse(A).b;
```

Taylor Series for $sin(x)$

```
> f:=(x,n)->convert(series(sin(x),x=0,n),polynom);
> plot({sin(x),f(x,2),f(x,4),f(x,6),f(x,8)},x=0..4);
```



Taylor Series for $sin(x)$

$$sin(x) = \sum_i (-1)^i \frac{1}{(2i+1)!} x^{2i+1} \qquad (cn < 1)$$

```
double mysin(double x, double target_absolute_precision) {
   if(x<0) return -mysin(-x);
   x=x-2.0*Pi*((int) (x/(2.0*Pi)));
   if(x>=Pi) return -mysin(2.0*Pi-x);
   if(x>=Pi/2) return mysin(x);
   double result=x;
   double term=x;
   for(int i=2;; i+=2) {
      term*=-x*x/i/(i+1);
      retult+=term;
      if(abs(term)<target_absolute_precision)
         return result;
   }
}
```

Taylor Series for $cos(x)$

$cos(x) = \sum_i (-1)^i \frac{1}{(2i)!} x^{2i}$ $\quad$ ($cn < 1$)

```
double mycos(double x, double target_absolute_precision) {
   if(x<0) return mycos(-x);
   x=x-2.0*Pi*((int) (x/(2.0*Pi)));
   if(x>=Pi) return mycos(2.0*Pi-x);
   if(x>=Pi/2) return -mycos(x);
   double result=1;
   double term=1;
   for(int i=1;; i+=2) {
      term*=-x*x/i/(i+1);
      retult+=term;
      if(abs(term)<target_relative_precision)
         return result;
   }
}
```

Taylor Series for $tan(x)$

$$tan(x) = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7 + O\left(x^9\right)$$

- $cn > 1$
- converges slowly
- converges only for $|x| < \pi/2$
- needs different approach:

```
double mytan(double x, double target_absolute_precision) {
   double d=mycos(x);
   if(d==0) throw "DivisionByZero";
   return mysin(x,target_absolute_precision)/
          mycos(x,target_absolute_precision);
   // precision issues with this definition!
}
```

Linear Algebra Algorithms

- Gauss Elimination: turn A in LU form
- Gauss-Jordan Elimination: turn $A, B = 1$ in $A' = 1, B' = A^{-1}$
- Cholesky Decomposition: turn A into LL^T form

where L is a lower triangular and U is upper triangular

Linear Algebra, Linear Systems

(1) $$x_0 + 2x_1 + 2x_2 = 3$$

(2) $$4x_0 + 4x_1 + 2x_2 = 6$$

(3) $$4x_0 + 6x_1 + 4x_2 = 10$$

equivalent to

(4) $$Ax = \begin{pmatrix} 1 & 2 & 2 \\ 4 & 4 & 2 \\ 4 & 6 & 4 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \begin{pmatrix} 3 \\ 6 \\ 10 \end{pmatrix} = b$$

Linear Algebra, Inversion

$$
\begin{aligned}
Ax &= b && && (5)\\
Ax &= Bb && B = 1 && (6)\\
\ldots &= \ldots && \text{Gauss-Jordan} && (7)\\
A'x &= B'b && A' = 1 && (8)\\
x &= B'b && B' = A^{-1} && (9)
\end{aligned}
$$

Linear Algebra, Gauss-Jordan

```
Matrix GaussJordan(Matrix A):
   make B identity matrix same size as A
   for every column c:
       find r>c such that abs(A(r,c)) is max
       if max is zero throw Exception
       swap rows r and c in both A and B
       coeff=A(c,c)
       for every column k:
           A(c,k)/=coeff
           B(c,k)/=coeff
       for every row r and r!=c:
           coeff=A(r,c)
           for every column k:
               A(r,k)-=coeff*A(c,k)
               B(r,k)-=coeff*B(c,k)
   return B ( the inverse of A)
```

Linear Algebra, Inversion

Theorem: If A is a square matrix and B=GaussJordan(A) does not throw an exception then B is the inverse of A ($AB = 1$) else A is a singular matrix ($\det A = 0$).

Linear Algebra, Norm

- $||x||_p \equiv (\sum_i abs(x_i)^p)^{\frac{1}{p}}$
- $||A||_p \equiv \text{max}_x ||Ax||_p / ||x||_p$
- $||x||_2 = \sqrt{\sum_i x_i^2}$ (preferred norm)
- $||A||_2$ difficult to compute but
- $||A||_2 \simeq ||A||_1$ and
- $||A||_1 = \text{max}_j \sum_i abs(A_{ij})$
- cond.num $\equiv ||A||_p * ||A^{-1}||_p$

Linear Algebra, Cholesky

```
Matrix Cholesky(Matrix A):
   if A is not symmetric throw Exception
   copy A into L
   for every column c:
       if L(c,c)<=0 throw Exception
       L(c,c)=sqrt(L(c,c))
       coeff=L(c,c)
       for every row r>c:
           L(r,c)/=coeff;
       for every column j>c:
           coeff=L(j,c)
           for every row r>c:
               L(r,j)-=coeff*L(r,c)
   for every column c:
       for every row r<c:
           L(r,c)=0
   return L
```

Linear Algebra, Cholesky

Theorem: If A is a symmetric matrix positive definite (for every x, $x^T Ax > 0$) and L=Cholesky(A) then L is a lower trinagular matrix such that LL^T is A. (If assumptions not met Cholewsky throws Exception)

Application: Covariance Matrix

$S_i(t)$ is stock price at time t for stock i.
$r_i(t) = \log(S_i(t+1)/S_i(t))$ are daily log-returns of stock i at time t.
$A_{ij} = \frac{1}{T}\sum_t r_i(t) r_j(t)$ is the correlation matrix

Problem: generate random vectors $\mathbf{r}$ with probability
$p(\mathbf{r}) \propto e^{-\frac{1}{2}\mathbf{r}^T A^{-1} \mathbf{r}}$

Notice: $p(\mathbf{r}) \propto e^{-\frac{1}{2}(L^{-1}\mathbf{r})^T (L^{-1}\mathbf{r})}$

Solution: $\mathbf{r} = L\mathbf{x}$ where x_i are normal random numbers, and
$L = Cholesky(A)$

Application: Markoviz Portfolio

$A_{ij} = \frac{1}{T} \sum_t r_i(t) r_j(t)$ is the correlation matrix, r_i is the average return of stock i, and $\bar{r}$ is the risk free interest.

Problem: Find the Optimal (Markoviz) Portfolio

Solution: $\mathbf{x} = A^{-1}(\mathbf{r} - \bar{r}\mathbf{1})$

Here $x_i/||\mathbf{x}||_2$ is the fractional amount of funds to be invested in the Markovitz portoflio.

Linear Least Squared

Problem: find vector $\mathbf{x}$ such that

$$||A\mathbf{x} - \mathbf{b}||_2$$

is minimum.

Solution 1: $\mathbf{x} = (A^T A)^{-1} A^T \mathbf{b}$

Solution 2: $\mathbf{x} = (L^T)^{-1} L^{-1} A^T \mathbf{b}$

where $L = Cholesky(A^T A)$.

Example, Polynomial Fitting

Problem: find x_i such that, given t_j and b_j,

$$x_0 + x_1 t_0^1 + x_2 t_0^2 + \ldots \quad = \quad b_0 \tag{10}$$

$$\ldots \quad \ldots \tag{11}$$

$$x_0 + x_1 t_m^1 + x_2 t_m^2 + \ldots \quad = \quad b_m \tag{12}$$

Solution: Solve Linear Least Squared with

$$A = \begin{pmatrix} 1 & t_0 & t_0^2 & \ldots \\ & & & \\ 1 & t_m & t_m^2 & \ldots \end{pmatrix} \tag{13}$$

Example, exponential Fitting

Problem: find x_i such that, given t_j, b_j, and c_j,

$$x_0 e^{c_0 t_0} + x_1 e^{c_1 t_0} + x_2 e^{c_2 t_0} + \ldots = b_0 \tag{14}$$

$$\ldots \quad \ldots \tag{15}$$

$$x_0 e^{c_0 t_m} + x_1 e^{c_1 t_m} + x_2 e^{c_2 t_m} + \ldots = b_m \tag{16}$$

Solution: Solve Linear Least Squared with

$$A = \begin{pmatrix} e^{c_0 t_0} & e^{c_1 t_0} & e^{c_2 t_0} & \ldots \\ & & & \\ e^{c_0 t_m} & e^{c_1 t_m} & e^{c_2 t_m} & \ldots \end{pmatrix} \tag{17}$$

Gram-Schmidt Ortho-gonalization

```
Matrix GramSchmidt(Matrix A):
   for every column i:
      r = norm2 of col i of matrix A
      if r==0: throw Exception
      for every col j>i:
         s=scalar product between col i and j of A
         for every row k:
            A(k,j)-=(s/r^2)*A(k,i)
   return A
```

Applications: R=Inverse(GramSchmidt(A))*A is triangular!

Gram-Schmidt Ortho-normalization

```
Matrix GramSchmidt2(Matrix A):
   for every column i:
      r = norm2 of col i of matrix A
      if r==0: throw Exception
      for every row k:
         A(k,i)/=r
      for every col j>i:
         s=scalar product between col i and j of A
         for every row k:
            A(k,j)-=s*A(k,i)
   return A
```

Non-Linear Solvers: Fixed Point

Example: $f(x) = x^2 - x - 3 = 0$

Solution: rewrite as $x = g(x) = x^2 - 3$ and solve iteratively

```
double FixedPoint(function g, float x_guess):
    x=x_guess
    x_old=x+precision
    while |x_old-x|<precision:
        if |g'(x)|>=1 throw Exception
        x_old=x
        x=g(x)
    return x
```

Non-Linear Solvers: Bisection

Example: $f(x) = 0$ and $sign(f(a))! = sign(f(b))$

```
double Bisection(function f, float a, float b):
    fa=f(a); if fa==0: return a
    fb=f(b); if fb==0: return b
    if fa*fb>0: raise Exception
    for k=0..max_steps:
        x=(a+b)/2
        fx=f(x)
        if |fx|<precision:
           return x
        else if fx*fa<0:
           b=x; fb=fx
        else if fx*fb<0:
           a=x; fa=fx
     raise Exception
```

Non-Linear Solvers: Newton Method

Example: $f(x) = 0$ Solution: use $f(x+h) \simeq f(x) + f'(x)h$

```
double Newton(function f, float x_guess):
    x=x_guess
    for k=0..max_steps:
        if |f(x)|<precision throw Exception
        x=x-f(x)/f'(x)
    raise Exception
```

Non-Linear Solvers: Newton+Bisection

Example: $f(x) = 0$

Solution: use $f(x+h) \simeq f(x) + f'(x)h$ and $f(x+h) = 0$ implies

$h = -f(x)/f'(x)$

Non-Linear Solvers: Newton+Bisection

```
double NewtonBisection(function f, float a, float b):
    fa=f(a); if fa==0: return a
    fb=f(b); if fb==0: return b
    if fa*fb>0: raise Exception
    f1x=0
    for k=0..max_steps:
        if f1x!=0: x=x-fx/f1x
        if f1x==0 or x<=a or x>=b: x=(a+b)/2
        fx=f(x)
        f1x=f'(x)
        if |fx|<precision:
           return x
        else if fx*fa<0:
           b=x; fb=fx
        else if fx*fb<0:
           a=x; fa=fx
    raise Exception
```

Non-Linear Solvers: Issues

- If there is a unique solution between a and b and function is continuous use Bisection
- ... if function is also differentiable use NewtonBisection.
- If function has a degenerate solution x, Bisection and NewtonBisection do not work. Try FixedPoint and Newton which may or may not converge.

1D Minimization

Solver	Optimizer
$f(x) = 0$	$f'(x) = 0$
one zero in domain	one extreme in domain
f	$\rightarrow f'$
f'	$\rightarrow f''$

1D Minimization: Newton Method

Approximate $f(x+h)$ with $f(x)+f'(x)h+f''(x)h^2/2$, and set its derivative to zero: $f'(x)+f''(x)h=0$ therefore $h=-f'(x)/f''(x)$.

```
double NewtonExtreme(function f, double x0):
    x=x0
    while true:
        x_old=x
        f1=(f(x+eps)-f(x))/eps
        f2=(f(x+eps)-2*f(x)+f(x-eps))/(eps*eps)
        x=x-f1/f2
        if abs(x-x_old)<PRECISION: return x
```

1D Minimization: Golden-Section Search

If f is continuous has a single minimum in $[a, b]$ then find x_1 and x_2 such that $a < x_1 < x_2 < b$, and if $f(x_1) < f(x_2)$ then minimum is in $[a, x_2]$ else minimum is in $[x_1, b]$. Iterate...

1D Minimization: Golden-Section Search

```
double GoldenSecion(function f, double a, double b, double t > 0.5):
    x1=a+(1-t)*(b-a)
    f1=f(x1)
    x2=a+t*(b-a)
    f2=f(x2)
    while abs(b-a)>PRECISION:
       if f1>f(a) or f1>f(b) or f2>f(a) or f2>f(b): throw Exception
       if f1>f2:
          a,x1,f1=x1,x2,f2
          x2=x1+(1.0-t)*(b-x1)
          f2=f(x2)
       else:
          a,x2,f2=x2,x1,f1
          x1=a+t*(x2-a)
          f1=f(x1)
    return b (or a)
```

1D Minimization: Golden-Section Search

The average running time is optimal if

$$t = (\sqrt{(5)} - 1)/2$$

N-D Minimization: Newton Method

Approximate $f(x+h)$ with $f(x) + \; f(x)h + h^T H(x)h/2$, and set its derivative to zero: $\bigtriangledown f(x) + H(x)h = 0$ therefore $h = -H(x)^{-1} \bigtriangledown f(x)$. ($H$ is Hessian)

```
vector NewtonExtreme(function f, vector x0):
    x=x0
    while true:
        x_old=x
        f1(x)=Gradient(f,x)
        f2(x)=Hessian(f,x)
        x=x-inverse(f2)*f1
        if abs(x-x_old)<PRECISION: break
```

N-D Minimization: Steepest Descent

Approximate $f(x+h)$ with $f(x) + \bigtriangledown f(x) h + h^T H(x) h/2$, and set its derivative to zero: $\bigtriangledown f(x) + H(x)h = 0$ therefore $h = -H(x)^{-1} \bigtriangledown f(x)$. ($H$ is Hessian)

```
vector SteepestDescent(function f, vector x0):
    x=x0
    while true:
        x_old=x
        f1=Gradient(f,x)
        minimize in alpha f(x-alpha*f1)
        x=x-alpha*f1
        if abs(x-x_old)<PRECISION: break
```

Can be very unstable