AMATH 242: Introduction to Computational Mathematics

Johnew Zhang

July 18, 2013

Contents

1 Introduction

Objective

- Analyze/Apply: Algorithms for problems in science, engineering, medicine, finance
- Identify possible sources of error
- Recognize conditioning of the data
- Estimate the stability of an algorithm
- Obtain a working knowledge of MATLAB: `http://www.mathworks.com/moler/index_ncm.html`. Cleve Moler, Original Author of MATLAB. NCM: A library of MATLAB functions. Website contains Moler's book.

What is numerical analysis ? Read article by Trephethen.

1.1 Problem (Saver)

Evaluate

$$p(x) = 2x^4 + 3x^3 - 3x^2 + 5x - 1$$

at $x = \frac{1}{2}$

1. $p(x) = 2 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} + 3 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} - 3 \cdot \frac{1}{2} \cdot \frac{1}{2} + 5 \cdot \frac{1}{2} - 1$ (10 multiplications and 4 additions)
2. Save $\frac{1}{2} \cdot \frac{1}{2} = (\frac{1}{2})^2$, $(\frac{1}{2})^2 \frac{1}{2} = (\frac{1}{2})^3$, $(\frac{1}{2})^3 \frac{1}{2} = (\frac{1}{2})^4$, then $p(x) = 2 \cdot (\frac{1}{2})^4 + 3 \cdot (\frac{1}{2})^3 - 3 \cdot (\frac{1}{2})^2 + 5 \cdot (\frac{1}{2}) - 1$ (7 multiplications and 4 additions)
3. Nested Multiplication

$$\begin{aligned} p(x) &= -1 + x(5 - 3x + 3x^2 + 2x^3) \\ &= -1 + x(5 + x(-3 + 3x + 2x^2)) \\ &= -1 + x(5 + x(-3 + x(3 + 2x))) \end{aligned}$$

   (4 multiplication and 4 additions)

   This is called Horner's rule. In general, any polynomial can be written in the nested shifted form $c_1 + (x - b_1)(c_2 + (x - b_2)(c_3 + (x - b_3)(c_4 + (x - b_4)(c_5)))) = c_1' + c_2'x + c_3'x^2 + c_4'x^3 + c_5'x^4$ for a degree-4 polynomial

   In MATLAB

```
>>function y=nest(c, x, b)
>>d=length(c)-1  %note need not specify d
>>y=c(d+1)
>>for i =d:-1:1
          y=y*(x-b(i))+c(i);
     end
```

Consider our example

```
>>b=zeros(4, 1) *
>>nest([-1 5 -3 3 2], 1/2, b)
>> ans
         y =1.25
* Can be included in the function by adding (before the loop):
IF nargrn< 3, b = zeros(d, 1);
Then call Function by
>> nest([-1 5 -3 3 2], 1/2)
Note: The nest command can be modified to take multiple arguments x by replicating
the multiple * by
y =y*(x -b(i))+c(i)
>>nest([-1 5 -3 3 2], [-2 -1 0 1 2])
>> ans
-1 5 -10 -1 6 53
x, y are now vectors of length = # evaluation points
```

2 Floating Point Numbers - Sources of errors

2.1 Binary Number

$\cdots, b_2, b_1, b_0, b_{-1}, b_{-2}, \cdots = \cdots + b_2 2^2 + b_1 2^1 + b_0 2^0 + b_{-1} 2^{-1} + b_{-2} + \cdots$

Convert 0.7 to binary:

$$\begin{aligned}
.7 \times 2 &= .4 + 1 \\
.4 \times 2 &= .8 + 0 \\
.8 \times 2 &= .6 + 1 \\
.6 \times 2 &= .2 + 1 \\
.2 \times 2 &= .4 + 0 \\
\implies (0.7)_{10} &= (0.1\overline{0110})_2
\end{aligned}$$

Convert $0.1\overline{0110}$ to decimal: $x = 0.1\overline{0110}$, $2^5 \cdot x = 10110.\overline{0110}$, $2 \times x = 1.\overline{0110}$.

$$\begin{aligned} 2^5 \cdot x - 2 \cdot x &= 10101 \\ 30x &= 21 \\ x &= 0.7 \end{aligned}$$

2.2 Floating point (IEEE double precision)

Sign bit, 52-bit mantissa (leading digit 1 is implies and omitted), 11-bit exponent (conversion: $-1022 \leq exp \leq 1023$)

Matlab returns "inf" for biggest float ($\pm 2^5 2 \cdot 2^{1023} \approx 10^{308}$). Underflow is complicated so we ignore it NaN exists too. Rounding: if 53rd digit is 0, truncate. If 1, add 1 to the 52ed digit. Exception: if all known digits to the right of 53rd are 0, add 52nd to itself.

For example: $fl(9.4) = 0001011001100110011001100110011001100 \cdots = 1.001\overline{0110} \times 2^3 = 9.4 - 0.4 \times 2^{-48} + 2^{-49}$ this is how much error there is.

```
>>format long \\ tell matlab to display 16 digits,
\\ this is cool because 10^15 < 2^52 < 10^16
>> x = 9.4
>> 6 = x - 9
>> z = y - 0.4
>> 3.0669072875470e-16
```

Decimal Examples of Roundoff

```
>> a = 1 + 1e-16; 1 +1e^-16 = 1 0.... 0 1, 17 decimal significant digits.
>>a - 1
>>and = 0; 17th digit got lost
>>a = 1+9e-16; also 17 decimal digits
>>a - 1
and = 8.8818e-16; 17th digit got fairly well represented
>>a =9.9 +1e-15; 16 decimal significant digits
>>a - 9.9
and 1.7764e-15; 16th digit got lost, 15th gets a bad representation
```

2.2.1 Roundoff Error

- Absolute: $|x - fl(x)|$
- Relative: $\frac{|x - fl(x)|}{|x|}$

- ε_{mach} = smallest number above $1 - 1$

  $\varepsilon_{mach} = 1.\underbrace{00\cdots0}_{\text{51 zeros}}1 - 1 = 0.\underbrace{0\cdots01}_{\text{52 digits}} = 2^{-52}$ Note:

  $$\frac{|x - fl(x)|}{|x|} = \frac{0.0\cdots x_{53}\cdots}{1.x_1\cdots x_{52}\cdots}$$

  $x = 1.x_1\cdots x_{52}\cdots \times 2^m$ and $fl(x) = 1.x_1\cdots x_{52} \times 2^m$.

  Note: If $fl(x)$ involved roundoff, then

  $$\frac{|x - fl(x)|}{|x|} \le \frac{1}{2}\varepsilon_{mach}$$

- Roundoff error accumulates after a sequence of operations. One particularly severe instance is catastrophic cancellation which occurs when two nearly equal numbers get subtracted.

  Example

  Evaluate $(x - 1.1)^{30}$ at $x = 1.2$

  $(1.2 - 1.1)^{30} = .1^{30} = 10^{-30}$.

  ```
  >>(1.2-1.1)^30
  >>ans 1.e-30

  Consider a different eval

  >>c=poly(ones(30, 1)*1.1) (*)
  e.g. poly([2, 3]) yields[1, -5, 6] since (x-2)(x -3) = x^2 -5x +6
  >>polyval(c, 1.2)
  evaluates poly c at x = 1.2
  >>ans -1.79e-5!
  ```

  What went wrong?

  Result of (*) is $[\cdots, \underbrace{-6\times10^8}_{\text{coefficient of } x^{15}}, \cdots]$ so this term is $-6\times10^8\times1.9^{15} \approx -10^{10}$. Rel error in this term is $\approx 10^{-16}$, i.e., ABS error is $\approx |-10^{10}|10^{-16} = 10^{-6}$ so ABS error in this one term exceeds the value of the sum which is 10^{-30} so final answer is overwhelmed by propagating roundoff.

3 Nonlinear Equations

4 Linear Systems

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2.099 & 6 \\ 5 & -1 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 3.901 \\ 6 \end{pmatrix}$$

Solve the system by gauss elimination. Matlab's command to solve the system $Ax = b$ is $A\backslash b$, which use elimination. Never $x = A^{-1}b$.

1st Elimination Step

- Multiply first equation $(-\frac{3}{10})$ (row and RHS) and subtract from second equation replace the second equation.
- multiply the first equation by 5/10, and subtract from the third equation. Replace the third equation.

Result is:

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.001 & 6 \\ 0 & 2.5 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6.001 \\ 2.5 \end{pmatrix}$$

Create a zero here to do this, should multiply the second equation by $-\frac{2.5}{0.001}$ and subtract from the third equation. Replace the third equation.

Denominators of Each Elimination Step 10 for the first step -0.001 for the second step are called pivots.

Note: that the points are the diagonal entries of A on the column we are working on (pivot column) to produce zeros below the diagonal.

Computation $-\frac{2.5}{0.001}$ + rest of alto leads to instability (more later)

Solution Switch the second and the third rows called pivoting. Now the pivot is 2.5 and the second elimination step is multiplying third equation by $-0.001/2.5$ subtract from the second and replace the third.

4.1 Gauss Elimination with Partial Pivoting (GEPP)

Partial: Row exchanges only (No columns)

For each column (except the last) seek the entry with largest absolute value on or below the diagonal position. Move to the diagonal position by swapping the two rows. For each row i below the pivotal row (row R), multiply pivotal row by $\frac{A_{iR}}{A_{RR}}$, called multiplier. Subtract from row i and replace row i. This creates zeros below the diagonal on column k.

```
function [L, U, p] = lutx(A)
[n, n] = size(A)
P=(1:n)
for k=1:n-1 % Loop over columns
% Find index of largest value below diagonal
[r, m] = max(abs(A(k:n, k)));
m=m+k-1;
if (A(m, k)~=0)
%swap pivot row
if (m~=k); If largest value already on diagonal, no need for swap
A([k, m], :)=A([m, k], :)
P([k, m])=P([m k])
end
% compute multiplier
A(k+1:n, k) =A(k+1:n)/A(k, k)
%Update the remainder of the matrix
A(k+1:n, k+1:n) = A(k+1:n, k+1:n) - A(k+1:n, k)*A(k, k+1:n)
```

4.1.1 GEPP (Matlab function lutx, $[L, U, P] = lutx(A)$)

```
k =1:n-1 % loop over columns
%Perform Pivoting
i = k+1:n
%Compute Multipliers
A(i, k) = A(i, k)/A(k, k)
j=k+1:n
%Elimination Update
A(i, j) = A(i, j) - A(i, k) *A(k, j)
end
```

Last time we did not write any loops over i and j because of parallel computation - vectorization. That Operates on all entries i, j at once. Loop over k is not vectorisable: each elimination step depends on the previous.

Separate Result

$L = tril(A, -1) + eye(n, n)$; where tril is the lower triangular matrix. Similarly, $U = triu(A)$; where trio is the upper triangular matrix. Recall, $p = (1, n)$ records row swaps: entries CDR to swapped rows are interchanged, P is A $n \times n$ matrix, whose rows are interchanged accordingly.

Recall from linear algebra, $PA = LU$ called the LU decomposition. Solution of

$$Ax = b$$

$$PAx = Pb$$
$$LUx = Pb$$
$$Ux = y$$
$$Ly = Pb$$

Note that we need P to operate on any given right hand side b. The last two equations are backward substitution and forward elimination respectively.

```
function x = forward(L, x) % Called after lute as y=forward(L,Pb)
[n, n]=size(L);
for k=1:n
j=1:k-1;
x(k)=(x(k)-L(k,j)*x(j))/L(k, k)
end
```

```
function x=backsubs(U,x) %in our notation x = backsubs(U, y)
for k=n:-1:1
y=k+1
x(k)=(x(k)-U(k, y)*x(y))/U(k, k)
end
```

Can look at GEPP as successive operations on matrix A, e.g., 3×3 example.

$$M_2 P_2 M_1 A = U$$

where M_k produces one elimination step contains negatives of the multiplies of the kWh elimination step.

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0.3 & 1 & 0 \\ -0.5 & 0 & 1 \end{pmatrix}$$

$$P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0.004 & 1 \end{pmatrix}$$

In general

$$M_{n-1} P_{n-1} \cdots M_k P_k \cdots M_1 P_1 A = U$$

Back to the example, note that

$$P_2 M_1 = N_1 P_2$$

with

$$N_1 = \begin{pmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0.3 & 0 & 1 \end{pmatrix}$$

So that GEPP is $M_2N_1P_2A = U$ or $PA = LU$. where $P = P_2$ and $L = (M_2N_1)^{-1} = N_1^{-1}M_2^{-1}$. To see how this work in general, consider the same when $n = 5$.

$$\begin{aligned} M_4 \qquad & \underbrace{P_4M_3} \qquad P_3M_2P_2M_1P_1A = U \\ {\scriptstyle N_3} \quad & \underbrace{P_4P_3M_2} \\ {\scriptstyle N_2} & \underbrace{P_4P_3P_2M_1}_{N_1P_4P_3P_2} \end{aligned}$$

Or

$$M_4N_3N_2N_1P_4P_3P_2P_1A = U$$

N_k is M_k with the negatives of the multipliers. And only those entries, not the 0's and 1's) swapped according to the permutation effected by all $P_{n-1} \cdots P_{k+1}$.

Note: $N_{n-1} = M_{n-1}$. Then write GEPP as

$$PA = L'U$$

where $P = P_4P_3P_2P_1$ and $L = (M_4N_3N_2N_1)^{-1}$. Note that P is unknown until the end of the computation.

4.2 Error Analysis

Consider 2×2 system of equations. Let x = exact solution($*$) and $\hat{x}$ = GEPP(0). The error will be $\|x - \hat{x}\|$. The residual is $\|A\hat{x} - b\|$.

Consider A different system of that two equations are very close to each other. Then the numerical solution is far from the exact solution but still very close to both equations, i.e., $A\hat{x} \approx b$, meaning the residual is small. Note that GEPP always gives small residual no matter how bad the solution is. Last case corresponds to a nearly singular matrix. Error is very large but the residual is small.

4.2.1 How Fast Is GEEP?

Bulk of the computation is

```
for k=1:n-1
      i=k+1:n
      j=k+1:n
      A(i, j) =A(i, j) - A(i, k)*A(k, j)
```

two operations (one multiplication and one addition) For each entry of the $(n-k)^2$ matrix, so total count is $FLOPS = 2(n-k)^2$ per k loop. Total flops are $\sum_{k=1}^{n-1} 2(n-k)^2 = 2\frac{(n-1)^3}{3} + O(n^2) = \frac{2}{3}n^3 + O(n^3)$.

4.2.2 Continue Discussion On GEPP Error

Vector Norms

$$\|x\|_1 = |x(1)| + \cdots + |x(n)|$$
$$\|x\|_2 = \sqrt{x(1)^2 + \cdots + x(n)^2}$$
$$\|x\|_\infty = \max\{|x(1)|, \cdots, |x(n)|\}$$

Theorem 1. 1. $\|x\| > 0 \iff x \neq 0$ and $\|x\| = 0 \iff x = 0$

2. $\|\alpha x\| = |\alpha|\|x\|$

3. $\|x + y\| \leq \|x\| + \|y\|$
Abstract definition of norm is any $F : \mathbb{R}^n \to \mathbb{R}$ satisfying 1, 2, 3 above.

4.3 Matrix Norms

$$\|A\|_1 = \max_{1 \leq j \leq n_{col}} \sum_{i=1}^{n_{row}} |A(i,j)| = \max_{\text{over columns}} \text{(1-norm of each colum)}$$

$$\|A\|_\infty = \max_{1 \leq i \leq n_{row}} \sum_{j=1}^{n_{col}} |A(i,j)| = \max_{\text{over rows}} \text{(1-norm of each row)}$$

$$\|A\|_F = \sqrt{\sum_{i=1}^{n_{row}} \sum_{j=1}^{n_{col}} A(i,j)^2} \text{ Frobenius norm}$$

2-Norm is computed through the SVD (singular value decomposition) of the matrix.

Theorem 2. All matrix norms defined above satisfy 1-3 and in addition, $\|Ax\|_{1 \text{ or } \infty} \leq \|A\|_1\|x\|_1$, $\|Ax\|_2 \leq \|A\|_F\|x\|_2$ and $\|AB\| \leq \|A\|\|B\|$

Theorem 3. All norms are related to each other by a factor which is A function of n.

Matlab: $norm(x, 1), norm(A, 1)$
Default for both is the 2-norm.

Definition. *Define condition number of a matrix A*

$$\kappa(A) = \|A\|\|A^{-1}\|$$

in some norm.
Properties:

1. $\kappa(A) \geq 1$

2. $\kappa(\alpha A) = \kappa(A)$. *If* $\kappa(A)$ *is close to 1, say* A *is well-conditioned, else ill-conditioned. If* A *is singular, say* $\kappa(A) = \infty$

Example:

1.

$$A = \begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.001 & 6 \end{pmatrix}$$

$$\|A\|_\infty = 17$$

$$A^{-1} = A\backslash I \text{ or inv(A)}$$

Check that $\|A^{-1}\|_\infty = 0.7331$

$$\kappa(A) = 17 \times 0.7331 = 12.4627$$

$$cond(A, inf) = 12.4627$$

A is a well conditioned matrix (more later as to when to be concerned)

2. A matrix's diagonals are all 2 but the last one is 2^{-n}Then $det(D) = 2^{n-1}2^n = 1/2$. For large n, the matrix is nearly singular (last row is close to 0).

$$\|D\|_1 = 2, \|D^{-1}\| = 2^n, \kappa(D) = 2^{n+1}$$

where $\kappa(D)$ is a good measure of near singular whereas $det(D)$ is not.

3. consider a Matrix D with all the diagonal 0.5. Then $\kappa(D) = \kappa(I) = 1$. This matrix is very-well conditioned. $d(D) = 0.5^n \to 0$. This indicates that for large n, the matrix is near singular (wrong conclusion).

$\kappa(A)$ describes the quality (conditioning) of the data.

Theorem 4. $A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n, Ax = b$. Let $\hat{A}, \hat{b}$ Perturbations of A, b of relative distance at most $\delta > 0$,

$$\frac{\|A - \hat{A}\|}{\|A\|} \leq \delta, \frac{\|b - \hat{b}\|}{\|b\|} \leq \delta$$

Let x solve $Ax = b$, $\hat{x}$ solve $\hat{A}\hat{x} = \hat{b}$. Then $\frac{\|x-\hat{x}\|}{\|x\|} \leq 4\delta\kappa(A)$. The following theorem is about GEEP. The question for GEEP is how good is the solution for a good set of data (A well conditioned matrix).

Theorem 5. Stability: result for GEEP (Wilingson):

$$\frac{\|x - x^{GEPP}\|}{\|x\|} \leq c(n)\kappa(A)\epsilon_{mach}$$

x is the exact solution. For most (almost all matrices, $c(n) = n^{\alpha}, \alpha < 1$, grows very slow with n few, specially constructed matrices exists such that $c(n) = 2^n$.

Note that with $c(n) \approx 1$, and a perfect matrix A, $\kappa(0 = 1$, error of algorithm is just ϵ_{mach}. GEPP is a phenomenally stable algorithm. A stable algorithm should give you results for every set of well-conditioned data. Suppose $c(n) \approx 1, \epsilon_{mach} = 10^{-16}, \kappa(A) = 10^{\alpha}$, Then wilkinson says

$$\frac{\|x - x^{GEPP}\|}{\|x\|} \leq 10^{-(16-\alpha)}$$

i.e., we lose α digits of accuracy ($16 - \alpha$ instead of 16) when we use GEPP. Notice the stark difference with catastrophic cancellation (bad result due to roundoff) which is due to an unstable algorithm. A similar type of "overpowering" (swamping) occurs in GE without PP. If pivot is small, then multiplier is large. This means multiplied equation overpowers. The one, it is subtracted from large multipliers are bad. GEPP guarantees multipliers are ≤ 1.

4.4 Iterative Methods for linear systems

4.4.1 Jacobi Method

$$A_{11}x_1^{(k+1)} + A_{12}x_2^{(k)} + \cdots + A_{1n}x_n^{(k)} = b_1$$

$$\vdots$$

$$A_{n1}x_1^{(k)} + A_{n2}x_2^{(k)} + \cdots + A_{nn}x_n^{(k)} = b_n$$

$$x_i^{(k+1)} = \frac{b_i - \sum_{i \neq j} A_{ij}x_j^{(k)}}{A_{ii}}$$

Gaus-Seidel usually converges faster (accelerated) by immediate use of new terms) and requires less storage and requires that updates be done. Successively whereas Jacobi can be run on A parallel computer. (updates independent of each other)

Definition. *We say that a sequence $\{P_n\}$ converges to the value P if and only if*

$$\lim_{n \to \infty} P_n = P$$

or equivalently,

$$\lim_{n \to \infty} |P_n - P| = 0$$

Definition. *Let sequence $\{P_n\}$, $\lim_{n\to\infty} P_n = P$, We say that $\{P_n\}$ converges to P with rate of convergence $O(\beta_n)$ if and only if $\exists\{\beta_n\}, \lim_{n\to\infty}\{\beta_n\} = 0$ and constant $\lambda > 0$ (independent of n) such that*

$$|P_n - P| \leq \lambda|\beta_n|$$

we write

$$P_n = P + O(\beta_n)$$

i.e. $O(\beta_n)$ is a references of how quickly the error $\epsilon_n = P_n - P$ approaches zero.

Example:

$$\lim_{n\to\infty} \frac{n+3}{n+7} = 1, \lim_{n\to\infty} \frac{2^n+3}{2^n+7} = 1$$

but

$$|\frac{n+3}{n+7} - 1| = \frac{4}{n+7} < 4\frac{1}{n}$$

$$|\frac{2^n+3}{2^n+7} - 1| = \frac{4}{2^n+7} < 4\frac{1}{2^n}$$

Hence the second sequence converges much faster.

Definition. *Let $\{P_n\}, \lim_{n\to\infty} P_n = P$. We say that $\{P_n\}$ converges to P with order of convergence α with asymptotic error constant λ if and only if $\exists\alpha, \lambda$ such that*

$$\lim_{n\to\infty} \frac{|P_{n+1} - P|}{|P_n - P|^\alpha} = \lim_{n\to\infty} \frac{|\epsilon_{n+1}|}{|\epsilon_n|^\alpha} = \lambda$$

i.e. rate of convergence examines individual error values $\epsilon_n = P_n - P$ (with the help of another sequence $\{\beta_n\} \to 0$ whereas order of convergence examines relation between successive error values.

Note that asymptotically:

$$\epsilon_{n+1}| \approx \lambda|\epsilon_n|^\alpha$$

i.e. Linear convergence $|\epsilon_{n+1}| \approx \lambda|\epsilon_n|$
Quadratic convergence $|\epsilon_{n+1}| \approx \lambda|\epsilon_n|^2$
Cubic convergence $|\epsilon_{n+1}| \approx \lambda|\epsilon_n|^3$

Numerical confirmation of, say, quadratic convergence is to evaluate $|\epsilon_n|/|\epsilon_{n-1}|^2$. And see if it approaches a constant as n increases. That would be the asymptotic error constant λ, If P is unknown, look at

$$\frac{|P_{n+} - P_n|}{|P_n - P_{n-1}|}$$

Consider the iteration $x_{k+1} = g(x_k)$ which solves the equation

$$g(x) = x$$

And is called a fixed point iteration.

4.4.2 Iteration

$g(x)$ can be a linear of nonlinear equation in one of serveral variables.

Theorem 6. Let $g : \mathbb{R} \to \mathbb{R}$ continuously differentiable (first derivative is continuous). and let x^* be the exact solution of $g(x) = x$. Then

1. If $|g'(x^*)| < 1$ and x_0 is sufficiently close to x^*, then the sequence $x_{k+1} = g(x_k)$ converges to x^* with
$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = |g'(x^*)|$$
i.e. convergence is linear can be better under special circumstances.

2. If $|g'(x^*) > 1$, then fixed point iteration will not converge unless by accident, $g(x_k) = x_k$ for some k.

3. If $|g'(x^*)| = 1$ may diverge or converge (but the convergences will be slow)

To solve $Ax = b$, consider splitting $A = M - N$, then consider $g(x) = M^{-1}Nx + M^{-1}b$. $g(x) = x$ is

$$M^{-1}Nx + M^{-1}b = x$$
$$Nx + b = Mx$$
$$(M - N)x = b$$
$$\text{or } Ax = b$$

Art is picking M and N. For example, Jacobi: $M + F$ diagonal entries of A and $N = -(A_L + A_U)$ and A_L is lower part of A and A_U is upper part of A. $A = D + A_L + A_U = M - N$ so fixed iteration $x_{k+1} = g(x_k)$ is $x_{k+1} = D^{-1}(b - (A_L + A_U)x_k)$ or

$$x_i^{k+1} = \text{ith entry of } k+1 \text{ iteration of vector } x = \frac{b_i - \sum_{i\neq j} A_{ij} x_j^{(k)}}{A_{ii}}$$

Indicial notion:Ax is $\sum_j A_{ij}x_j$. Then AB is $A_{i|}$

Fixed point iteration for linear systems

$$Ax = b$$
$$x_{k+1} = M^{-1}Nx_k + M^{-1}b$$

M, N splitting matrices, $A = M - N$.

4.5 Generalized Theorem of Previous Lecture to this case of multi-dimension linear problem

Theorem 7. For the above splitting iteration and a nonsingular matrix A, $\lim_{k\to\infty} x_k = A^{-1}b$ provided

$$\|M^{-1}N\} < 1$$

for some induced matrix norm

Recall an induced matrix norm is defined through a vector norm as

$$\|A\| = sup_{x\neq 0}\frac{\|Ax\|}{\|x\|}$$

Does this loo anything like the singular var theorem?

Proof. Let $x^* = A^{-1}b$. To show convergence, need to establish a relationship between $x_{k+1} - x^*$ and $x_k - x^*$ we are looking for something like

$$\|x_{k+1} - x^*\| \leq \lambda \|x_k x^*\|^\alpha$$

we can show that $x_{k+1} - x^* = M^{-1}N(x_k - x^*)$

$$\begin{aligned}
x_{k+1}M^{-1}Nx_k + M^{-1}b \\
(M-N)x^*b \\
(I - M^{-1}N)x^* &= M^{-1}b \\
x_{k+1} - x^* &= M^{-1}Nx_k + (I - M^{-1}N)x^* - x^* \\
&= M^{-1}N(x_k - x^*)\|x_{k+1} - x^*\| &= \|M^{-1}N(x_k - x^*)\|
\end{aligned}$$

Recall that induce dorm have the property

$$\|Ax\| \leq \|A\|\|x\|$$

This implies

$$\|x_{k+1} - x^*\| \leq \|M^{-1}N\|\|x_k - x^*\|$$

It follows that for convergence. It is sufficient that the asymptotic error constant $\lambda = \|M^{-1}N\| < 1$ and the order of convergence is linear ($\alpha = 1$). □

Remark

1. No dependence on starting point x_0. Nonlinear iteration method ($g(x)$ a non linear function of x) usually require x_0 in some neighborhood of x^*

2. If $\|\|$ is not an induced norm but it dominates an induced norm, then theorem is still true. For example, $\|A\|_F \geq \|A\|_2$ so suffices to show $\|M^{-1}N\|_F < 1 \implies \|M^{-1}N\|_2 \leq 1$.

3. A more relaxed condition is provided by the following theorem

   Theorem 8. The a voce splitting iteration converges if $\rho(M^{-1}N) < 1$ where $\rho(M^{-1}N)$ is the spectral radius of $M^{-1}N$, but it is harder to prove and may be harder to prove in practice.

Definition. *Spectral radius of a matrix is*

$$\rho(A) = \max\{|\lambda| : \lambda \in \lambda(A)\}$$

$\lambda(A)$ = *spectrum* = *set of all eigenvalues of* A.

Note:

$$\rho(A) \leq \|A\|$$

where $\|A\|$ any induced norm of A hence a more relaxed condition (easier to satisfy).

For Jacobi and Gauss-Seidel, if A is SROD, then these methods will converges.

Definition. $A \in \mathbb{R}^{n\times n}$ *is strictly row diagonally dominant (SROD) if*

$$|A_{ii}| > \sum_{j\neq i} |A_{ij}|, \forall i$$

.

Proof.

$$\begin{aligned}
M &= D \\
N &= -(A_U + A_L) \\
M^{-1}N &= -D^{-1}(A_U + A_L) \\
(M^{-1}N)_{ij} &= \begin{cases} 0 & i = j \\ -\frac{A_{ij}}{A_{ii}} & i \neq j \end{cases} \\
\|M^{-1}N\|_\infty &= \max(\sum_j |\frac{A_{ij}}{A_{ii}}|) = \max(\frac{1}{|A_{ii}|}\sum |A_{ij}|)
\end{aligned}$$

But A srod implies $\sum_j |A_{ij}| < |A_{ii}|, \forall i$. This implies $\frac{1}{|A_{ii}|}\sum_j |A_{ij}| < 1, \forall i$ so also the max. This implies $\|M^{-1}N\|_\infty < 1$ which means convergence □

Note: sets of rate and order of convergence given earlier. Assume a convergent sequence.

4.6 Nonlinear Equations

1. Fixed point iteration (As seen already)

   Example:

   $f(x) = \cos x - x$.

   (a) $g(x) = \cos x$ since $g(x) = x$, i.e., $\cos x = x$ means $f(x) = 0$. $g'(x) = -\sin x$, $g'(x^*) \approx -0.674$ so convergence is expected and the convergence will be linear

   (b) $g(x) = \cos^{-1}(x)$, $g(x) = x \iff \cos^{-1} x = x \iff x = \cos x \iff f(x) = 0$ so the fixed point iteration solves $f(x) = 0$. $g'(x^*) \approx -1.48$ so divergence is expected.

   (c) $g(x) = x + \frac{\cos x - x}{\sin x + 1}$ and $g(x) = x \iff \frac{\cos x - x}{\sin x + 1} = 0 \iff \cos x = x \iff f(x) = 0$.

   Method is $x_{k+1} = g(x_k)$ and $x_{k+1} = x_k + \frac{\cos x_k - 1}{\sin x_k + 1}$. $g'(x) = 0$ which is called super linear convergence.

   Fact: quadratic convergence is a special case of super linear. Note that this fixed point iteration is newton's method for $f(x) = \cos x - x = 0$. recall newton's method

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{\cos x_k - x_k}{-\sin x_k - 1}$$

4.6.1 Convergence Theorem for Newton's Method

Let f, f', f'' defined in $[x^* - \mu, x^* + \mu]$ where $f(x^*) = 0, \mu > 0$, and $\rho, \delta > 0$ such that

$$|f'(x^*)| \geq \rho, \text{ f not too flat}$$

$$|f''(x)| \leq \delta, \text{ f not too oscillatory}$$

$$\mu = \frac{\rho}{2\delta}$$

1. If $x_0 \in I$, then $x_k \in I, \forall k = 1, 2, \cdots$. I = window of convergence
2. $|x_{k+1} - x^*| \leq \frac{\delta}{\rho}|x_k - x^*|^2$. Weaker definition of quadratic convergence.

   Note This does not implies convergence.
3. $|x_{k+1} - x^*| \leq \frac{1}{2}|x_k - x^*|$ convergence.

   Recall definition of quadratic convergence was

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} = \lambda$$

Convergence window $I = [x^* - \mu, x^* + \mu]$, $\mu = \frac{\rho}{2\delta}$ where ρ = lower bound on slope and δ = upper bound on curvature

Notes on convergence

1. Convergence $x_k \to x^*$ means $\forall \delta > 0, \exists n(\delta)$, such that $\forall k \geq n, \|x_k - x^*\| \leq \delta$.

2. $|x_{k+1} - x^*| \leq c|x_k - x^*|, c < 1$. Weak definition of linear convergence. This implies $|x_k - x^*| \leq c^k|x_0 - x^*| = c_0c^k$ (yet another definition of linear convergence, i.e. linear convergence means $|x_k - x^*| \leq c_0c^k$.

   Let $c_0c^k \leq \delta$,

$$\begin{aligned} \ln c_0 + \ln c &\leq \ln \delta \\ k \ln c &\leq \ln \delta / c_0 \\ k &\geq \frac{\ln \delta / c_0}{\ln c} \end{aligned}$$

   i.e. $\forall \delta \geq 0, \exists n = \lceil \frac{\ln \delta / c_0}{\ln c} \rceil$., $\forall k \geq n, c_0c^k \leq \delta \implies |x_k - x^*| \leq \delta$ i.e. sequence converges.

   Earlier definition

$$\lim_{n \to \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} < c, c < 1$$

   is the strongest definition of linear convergence and gives asymptotic result $\epsilon_{n+1}| \approx c|\epsilon_n|^2$. does not require $c < 1$ and does not imply convergence.

3. $|x_k - x^*| \leq c_0c^k \implies \log|x_k - x^*| \leq \log c_0 + k \log c$. Hence $\log|\epsilon_k| < \log c_0 + k \log c$. i.e., asymptotically $\log|\epsilon_k| \approx k|\log c|$ i.e., each iteration contributes about $|\log c|$ decimal ($\log_{10}$) or binary ($\log_2$) digits, for example,

$$\log_p |\epsilon_k| = -6 \implies |\epsilon_k| = 10^{-6}$$

   i.e. 5 correct digits.

   Characteristically, an iteration that converges linearly in the sense of 1 or 2 will gain $\sim |\log c|$ correct digits per iteration.

4. In the case of fixed point iteration these inequalities are typically tight,

$$x_k - x^* \approx c_0c^k, c < 1$$

$$\frac{x_{k+1} - x_k}{x_k - x_{k-1}} \approx c$$

5. Quadratic convergence

$$|x_{k+1} - x^*| \leq c|x_k - x^*|^2$$

$$\log|\epsilon_{n+1}| \leq \log c + 2\log|\epsilon_k|$$

   Recall $\log|\epsilon_k|$number of correct digits in kth iteration

   i.e. Number of correct digits double each iteration.

4.7 Bisection Method

Based on the intermediate value theorem: If $f \in [a, b]$ and $f(a)f(b) \leq 0$ and $f(x)$ continuous, then $\exists x \in [a, b]$ such that $f(x) = 0$.

Algorithm:

```
while b -a >= delta
     m = (a+b)/2;
     if f(a)*f(m) <=0
        b = m;
     else
        a = m;
     end
end
```

```
If delta < max(|a|, |b|)*epsilon
May never terminate due to roundoff
```

$$b = 1.0 \cdots 1 \times 2^{exp}$$

$$a = 1.0 \cdots 0 \times 2^{exp}$$

so $\frac{a+b}{2} \approx a$ or b.

Note on code: can improve on multiple function evaluations.

MATLAB handles

Above code if literally used in MatLab is function specific, e.g, bisef.m. Alternatively

```
function x = bisection f, a, b, delta)
<body as earlier>
```

In this case f must be defined as a handle. f can be built in function or a function that you wrote.

For example:

$$f = @sin;$$

$$f = @myfunc;$$

$$f = @(x)(exp(x) - 2);$$

Note x is a dummy variable, i.e., $x = 2$. $f(@(x)(exp(x) - 2)$ will not evaluate the function at $x = 2$. Need to specify all but one variable, e.g., $x = -1 : 0- : 1$ and $y = \cdots$. $f = @(u)(polyinterp(x, y, u)$ and $p =$ [coeffs of polynomial],$f = @(p, x)(polyval(p, x))$ is valid handle.

Theorem 9. There is a root x^* to f in $[a^{(k)}, b^{(k)}]$ such that

$$|x^{(k)} - x^*| \leq \frac{b^{(0)} - a^{(0)}}{2^{k+1}}$$

where $x^{(k)} = m^{(k)}$. i.e. bisection method converges linearly in the sense that

$$|x^{(R)} - x^*| \leq c_0 c^k$$

with $c = \frac{1}{2}$ and $c_0 = (b^{(0)} - a^{(0)})/2$.

Recall above inequality is a weak definition of linear convergence.

4.8 Systems of nonlinear equations

Given $f : \mathbb{R}^n \to \mathbb{R}^n$ find x^* such that $f(x^*) = 0$. Applications: any steady state problem. Preferred algorithm: Newton's method

Theorem 10. Taylor theorem: $f : \mathbb{R}^s n \to \mathbb{R}^m, f(x+h) = f(x) + J(x)h + O(\|h\|^2)$. where

$$J(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

is the Jacobean.

Given $x^{(k)}$ obtain $x^{(k+1)}$ from linear approximation at $x^{(k)}$

$$f(x_k + h) \approx l(x_k + h) = f(x_k) + J(x_k)h$$

Let x_{k+1} be the root of $l(x_k + h) = l(x_{k+1}) = 0$.

$$f(x_k) + J(x_k)(x_{k+1} - x_k) = 0 \implies x_{k+1} = x_k - J^{-1}(x_k)f(x_k)$$

Midterm is up to bisection

4.9 Multivariate Newton

Note:

```
bisec.m
```

```
deltamin= eps*max(abs(a), abs(b))
```

Recall that the closest we can represent a number x on the computer is $x \times eps \times |x|$.

$$eps = \frac{x - fl(x)}{|x|}$$

$$x_{k+1} = x_k - J_k^{-1}(x_k)f(x_k)$$

Solve linear system $J(x_k)x_{k+1} = f(x_k)$. At each iteration, i.e., $O(n^3)$ flops per iteration often evaluate $J(x)$ not at each iteration but not every so many iterations (Quasi-Newton) also secant method. Obtain $J(x)$

1. Closed form
2. finite differences
3. Automatic Differentiation

Application: optimization (unconstrained local minima) .

Definition. *$x \in \mathbb{R}^n$ is a local minimizer of f if and only if $\exists r > 0$ such that $\|y - x\| \leq r \implies f(x) \leq f(y)$.*

Theorem 11. Suppose: $f : \mathbb{R}^n \to \mathbb{R}$ is continuous to second derivative and x^* satisfies

$$\nabla f(x^*) = 0$$

$$\nabla^2 f(x^*)$$

is positive definite. Then x^* is a local minimizer of f.

Recall gradient

$$\nabla f = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n})^T$$

Hessian (or Jacobean of ∇f)

$$\nabla^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Theorem 12. If x^* is a local minimizer, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite (i.e. $x^T \nabla^2 f x \geq 0$ for all x).

4.10 Newton's Method for Optimizer

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

Check that $\nabla^2 f$ is positive definite at each iteration! (otherwise newton may converge to a non-minimum).

Theorem 13. Symmetric, real matrices are always diagonalizable, have real eigenvalues & eigenvectors and eigenvalues can be made orthogonal to each other.

Power Method

$$\tilde{x}^{(k)} = Ax^{(k-1)}$$

normalization to avoid overflow or underflow.

Does this converge?

Recall: symmetric real matrices have real eigenvalues and eigenvectors can be taken real and orthonormal. For example, in case of a multiple eigenvalues, eigenvectors will span a space and can be made orthogonal in that space, orthonormal: $v_i v_j = \begin{cases} 0 & z \neq j \\ 1 & z \neq j \end{cases}$. $\frac{v_i}{\|v_i\|}, \frac{v_j}{\|v_j\|}$, orthonormal (if v_i, v_j orthogonal).

For power method, requires $|\lambda_i| \neq |\lambda_j|$.

$$\tilde{x}^{(k)} = Ax^{(k-1)}$$

$$x^{(k)} = \frac{\tilde{x}^{(k)}}{\|\tilde{x}^{(k)}\|}$$

suppose that A is diagonalisable.

$$D = \begin{pmatrix} \lambda_1 & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \lambda_n \end{pmatrix}, |\lambda_1| \neq |\lambda_j|, \forall j = 2, \cdots, n$$

Say $x^{(0)} = \Phi y^{(0)}$ some $y^{(0)}$ (Recall diagonalisation $A = \Phi D \Phi^{-1}, \Phi \in C^{n \times n}$, invertible and $D \in C^{n \times n}$ linearly independent eigenvectors. Apply the algorithm:

$$\tilde{x}^{(1)} = Ax^{(0)} = \Phi D \Phi^{-1} y^{(0)} = \Phi D y^{(0)}$$

$$x^{(1)} = \frac{\tilde{x}^{(1)}}{\|\tilde{x}^{(1)}\|} = \frac{1}{\|\tilde{x}^{(1)}\|} \Phi D y^{(0)}$$

$$\tilde{x}^{(2)} = Ax^{(1)} = \|\tilde{x}\|^{-1} \Phi D \Phi^{-1} \Phi D y^{(0)} = \|\tilde{x}^{(1)}\|^{-1} \Phi D^2 y^{(0)}$$

$$x^{(2)} = \tilde{x}^{(2)} / \|\tilde{x}^{(2)}\| = \|\tilde{x}^{(1)}\|^{-1} \|\tilde{x}^{(2)}\|^{-1} \Phi D^2 y^{(0)}$$

$$x^{(k)} = (\prod_{i=1}^{k} \|\tilde{x}^{(i)}\|)^{-1} \Phi D^k y^{(0)} = (\prod_{i=1}^{k} \|\tilde{x}^{(i)}\|)^{-1} \lambda_1^k \Phi \begin{pmatrix} 1 & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \lambda_n/\lambda_1 \end{pmatrix}^k y^{(0)}$$

Note:

$$\begin{pmatrix} 1 & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \lambda_n/\lambda_1 \end{pmatrix}^k \to \begin{pmatrix} 1 & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & 0 \end{pmatrix}$$

Then

$$\Phi \begin{pmatrix} 1 & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \lambda_n/\lambda_1 \end{pmatrix}^k = \Phi[:, 1] y^{(0)}$$

means $v^{(k)}$ has a finite limit, say v^*. Since $\|x^{(k)}\| = 1$, $\|x^{(k)}\| = \|\beta^{(k)} v^{(k)}\| = 1 \implies |\beta^{(k)}| = \frac{1}{\|v^{(k)}\|}$, $(\prod_{i=1}^{k} \|\tilde{x}^{(i)}\|)^{-1} \lambda_1 = \beta^{(k)}$ Hence $|\beta^{(k)}| \to \frac{1}{\|v^*\|}$. This implies $x^{(k)} \to <$ some scalar $>$ $\Phi[:, 1] y^{(0)}$. i.e. $x^{(k)}$ converges to the first column of Φ, which is the eigenvector corresponding to λ_1, called the dominant eigenvector. Must have $y^{(0)} \neq 0$, which cannot be guaranteed since Φ is unknown at the beginning of the algorithm. this ($y_1^{(0)} = 0$ is a low probable event. If $y_1^{(0)} = 0$, then power method converges to eigenvector of the second largest eigenvalue.

Inverse Power Method

Same, except $\tilde{x}^{(k)} = A^{-1} x^{(k-1)}$. Converges to the eigenvector corresponding to the smallest eigenvalue of A. Recall that A, A^{-1} have same eigenvectors and reciprocal eigenvalues. Implementation: use lute to obtain LU decomposition of A ($PA = LU$). Then use that in each step to save, $A\tilde{x}^{(k)} = x^{(k-1)}$.

Inverse Shifted Power Method

$$A = \Phi D \Phi^{-1}$$

$$A - \mu I = \Phi (D - \mu I) \Phi^{-1}$$

$$(A - \mu I)^{-1} = \Phi (D - \mu I)^{-1} \Phi^{-1}$$

so inverse power method on $A - \mu I$ converges to eigenvectors whose eigenvalue $\frac{1}{\lambda_i - \mu}$ is largest, i.e., closest to μ.

First Discretize and then find y that minimize the functional

$$F = \int_0^{x_f} \sqrt{\frac{1 + (y')^2}{-ky} dx}$$

Functional: A scalar function for a function $y(x)$. Discretize such that unknown is a vector y instead of a function $y(x)$.

4.11 Quadrature (Numerical Integration)

Compute an approximation to $\int_a^b f(x)dx$.

- Newton-Coates
- Gauss Quadrature

4.11.1 Newton-Coates

Choose m evenly spaced points $x_1, \cdots, x_m$ in $[a, b]$, i.e.,

$$x_i = a + \frac{i-1}{m-1}(b-a), i = 1, \cdots, m$$

$$(NC)_m f_a^b = (b-a)\sum_{i=1}^{m} w_i^{(m)} f(x_i)$$

where $w_i^{(m)}$ = weights at m points

Obtained by requiring that polynomials up to order m^{-1} be integrated exactly, for example, $m = 2$ should integrate exactly $f(x) = 1$,

$$f(x) = c_1 + c_2 x$$

$$x_1 = a$$

$$x_2 = b$$

$$(NC)_2(1)_a^b = (b-a)(w_1 + w_2) = \int_a^b 1dx = (b-a)$$

$$\begin{aligned}(NC)_2 f(x)_a^b &= (b-a)[w_1(c_1 + c_2 a) + w_2(c_1 + c_2 b)] \\ &= \int_a^b (c_1 + c_2 x)dx = c_1(b-a) + \frac{c_2}{2}(b^2 - a^2) = \frac{1}{2}(b^2 - a^2)\end{aligned}$$

$$(NC)_2(x)_a^b = (b-a)(w_1 a + w_2 b) = \frac{1}{2}(b^2 - a^2) = \int_a^b xdx$$

Then we can solve above to get

$$w_1 + w_2 = 1$$

$$aw_1 + bw_2 = \frac{a+b}{2}$$

Unique solution is

$$w_1 = w_2 = \frac{1}{2}$$

so that two points NC integration is

$$(NC)_2 f_a^b = (b-a)(f(a)+f(b))$$

called trapezoidal rule.

If $m = 3$: $x_1 = a, x_2 = a + \frac{1}{2}(b-a) = \frac{b+a}{2}, x_3 = b$

Require 1, x, x^2 integrate exactly $\implies w_1 = \frac{1}{6}, w_2 = \frac{2}{3}, w_3 = \frac{1}{6}$.

$$(NC)_3 f_a^b = (b-a)(\frac{1}{6}f(a) + \frac{2}{3}f(\frac{a+b}{2}) + \frac{1}{6}f(b))$$

Using Simpson 's Rule

4.11.2 Gauss Quadrature

$$G_m f_a^b = (b-a)\sum_{i=1}^{m} w_i^{(m)} f(a + (b-a)\xi_i^{(m)})$$

$0 < \xi_i < 1$ interior points. Some texts take $-1 < \xi_i < 1$

$m = 1$:

$$G_1 f_a^b = (b-a) w_1 f(a + (b-a)\xi_i)$$

$$G_1(1)_a^b = (b-a) w_1 = (b-a)$$

$$G_1(x)_a^b = (b-a) w_1 (a + (b-a)\xi_1)$$

Hence $\xi_1 = \frac{1}{2}$

If $m = 2$:

$$G_2(f)_a^b = (b-a)[w_1 f(a + (b-a)\xi_1) + w_2 f(a + (b-a)\xi_2)]$$

Four unknowns: w_1, w_2, ξ_1, ξ_2

Integrate exactly $1, x, x^2, x^3$. Hence $w_1 = \frac{1}{2} = w_2$, $\xi_1 = \frac{1}{2} = \frac{1}{\sqrt{12}}$ and $\xi_2 = \frac{1}{2} + \frac{1}{\sqrt{12}}$.

For any m, make Gauss exactly for polynomials up to order $2m-1$. yields $2m$ equations in unknown. $w_i \geq 0$ and $\xi_i \in (0,1)$ always gauss is more accurate than NC.

4.11.3 Compound Integration (Mesh Refinement)

Break interval into pieces. Perform NC or Gauss in each piece

$$\int_0^1 f(x)dx \approx G_2(f)_0^{3/4} + G_2(f)_{3/4}^1$$

Usually choose m first. Accuracy increases as number of segments increases: Theorems.

$$t = \sqrt{\frac{1+(y')^2}{-ky}}$$

time to reach $(x_f, y_f). From \mathrm{v}_x = \frac{dx}{dt}$. Find t_{min} by minimizing integral (functional) with respect to $y(x)$. Discretize, i.e., evaluate integral by using $N-1$ subintervals and 1 quadrature points and the midpoint rule. Minimize integral by $\sum\sum w \times$ values at integral points (Here one integral point) with respect to vector y.

Vector y = value of $y(x)$ at integral point. Usually, in practice pick type of integration (NC or Gauss And m) First, then take subdivisions. Above sum $f(y) : \mathbb{R}^N \to \mathbb{R}$ multivariate newton for optimization.

4.11.4 Multiple Newton for Optimization

$$x_{k+1} = x_k - (\nabla^2 g(x_k))^{-1} \nabla g(x_k)$$

Here $g(y_i) = \sqrt{\frac{1+(y')_i^2}{-ky_i}}$.Sample results for grad, Hessian given N, y in brachisto.mat (load brachisto). Note: NC approximates integrand by a $m-1$ degree polynomial evaluates integral of that polynomial.The better the integrand is approximated by a polynomial the better NC of Gauss will approximate the integral Gauss is generally faster than NC.

4.11.5 Accuracy of NC (Truncation Error)

Theorem 14. $\int_a^b f(x)dx - NC_m(f)_a^b = c_m f^{(s+1)}(\xi)(\frac{b-a}{m-1})^{S+2}, S = \begin{cases} m-1 & m \text{ even} \\ m & m \text{ odd} \end{cases}$

Proof is based on interpolation theory.

Above theorem says method is converging for $b-a \to 0$ (point backs compound integration and, at least for polynomials, method converges with increasing m (not true for all functions. For $m = odd$. Method is exact for polynomial up to order m.

Gauss: $|G_m(f)_a^b - \int_a^b f(x)dx| \le \frac{(b-a)^{2m+1}(m!)^4}{(2m+1)((2m)!)^3} M_{2m}$ where M_{2m} is a constant that bounds $|f^{(2m)}(x)|$ on $[a, b]$. GS is faster (i.e. more accurate for given $[a, b]$ - usually preferred).

5 Interpolation

Side note: adaptive quadrature: the algorithm estimates the error for a stack of intervals initiated to $[a, b]$, for example, by applying quadrature of order m and $m+1$ in each interval. If difference if below tolerance, proceed. Otherwise subdivide intervals.

Given discrete data points, find a continuous function, called the interplant, usually a polynomial, that passes through them.

Theorem 15. Given $(x_1, y_1), \cdots, (x_n, y_n)$ pairs of real numbers such that all $x_i's$ are distinct. Then $\exists$ unique polynomial of degree at most $n-1$, $p(x) = a_{n-1}x^{n-1} + \cdots + a_0$, such that $p(x)$ interpolates the data.

The naive way is to solve n equations and n unknowns

$$a_0 + a_1 x_1 + \cdots + a_{n-1} x_1^{n-1} = y_1$$

$$\vdots$$

$$a_0 + a_1 x_n + \cdots + a_{n-1} x_n^{n-1} = y_n$$

In Matrix Form
$$\begin{pmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ & \cdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

where the matrix is Vandermonde matrix.The V matrix is nonsingular but is ill-conditioned. Also, "naive" method required $O(n^3)$ operations (flops).

5.1 Lagrange Form of Polynomial

$$p(x) = \sum_k (\prod_{j \neq k} \frac{x - x_j}{x_k - x_j}) y_k$$

Theorem 16. Main accuracy theorem (Interpolation): Let I be an interval containing $x_1 < x_2 < \cdots < x_n$, and $f(x)$ an n-times differentiable function defined on I. Let $p(x)$ be the polynomial interplant of $(x_1, f(x_1)), \cdots, (x_n, f(x_n))$ of degree $\leq n-1$, i.e., $f(x_i) = p(x_i)$. Then, for any $x \in I$,

$$f(x) - p(x) = \frac{f^{(n)}(\xi)}{n!}(x - x_1) \cdots (x - x_n)$$

where ξ is some point in I. It may happen that $f^{(n)}(\xi)$ grows with n faster $n!$. In that case, the accuracy diminishes with growing n (order of interpolation), e.g., $f(x) = \frac{1}{1+25x^2}$.

Solution is piecewise polynomial functions.

Simplest: linear

Given $(x_1, y_1), \cdots, (x_n, y_n), x_1 < x_2 < \cdots < x_n$. Define

$$L(x) = y_i \frac{x_{i+1} - x}{x_{i+1} - x_i} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i}, x \in [x_i, x_{i+1}]$$

Nonlinear equations: compare $k, k+1$ iterates to figure tout the rate of convergence.

Adaptive compound:

Quadrature: compare order m, integration. If within tolerance interval is good else subdivide this called a posteriori error estimation: solve the problem partly in order to make decision. A priori: based on function properties. hard most code do a posteriori.

5.1.1 Piecewise linear interpolation

1. $L(x)$ (the interpolant) restricted to $[x_i, x_{i+1}]$ is a linear function.
2. L is continuous.
3. $L(x_i) = y_i, i = 1 : n$.

Theorem 17. Given $(x_1, y_1), \cdots, (x_n, y_n)$ pairs, $x_1 < \cdots < x_n$, $L(x)$ is the unique function with properties 1-3.

x_i's are called break points or knots. Note: $L(x)$ is generally non differentiable at knots.

$$L_i(x) = y_i \frac{x_{i+1} - x}{x_{i+1} - x_i} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i}, x \in [x_i, x_{i+1}]$$

which is in the form of a lagrange interpolant or

$$L_i(x) = y_i + (x - x_i) \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = y_i + s\delta_i$$

where $s = x - x_i, \delta_i =$ slope.

Theorem 18. Assume $f(x)$ twice differentiable and $L(x)$ is piecewise linear interplant to $(x_1, f(x_1)), \cdots, (x_n, f(x_n))$ with $x_1 < x_2 < \cdots < x_n$. Then $\forall x \in [x_1, x_n]$.

$$|f(x) - L(x)| \leq \max(\frac{(x_{i+1} - x_i)^2}{8} \max |f''(\xi)|)$$

Proof. Apply main theorem in each interval:

$$|f(x) - L_i(x)| = \frac{|f''(\xi)|}{2} |x - x_i||x - x_{i+1}|$$

Note $max(|x-x_i||x-x_{i+1}|) = \frac{(x_{i+1}-x_i)^2}{4}$ occurs at midpoint.

$$|f(x) - L_i(x)| \leq \max \frac{f''(\xi)|}{8}(x_{i+1}-x_i)^2$$

Then take max over all intervals. $\square$

Note:

1. Accuracy goes up with n (bound becomes smaller for smaller intervals)
2. Succest Adaptive interpolation: to reduce $|f(x) - L(x)|$ most effectively (i.e. using smallest number of knots) use more knots where $|f''(x)|$ is large. $|f''(x)|$ may be unknown may need some kind of a posteriori error estimate, for example, finite differences.

Disadvantages

1. Low accuracy $(x_{i+1}-x_i)^2$.
2. visible knots.

5.1.2 Cubic Hermite Interpolate

Data is $(x_1, y_1, d_1), \cdots, (x_n, y_n, d_n)$ $x_1 < \cdots < x_n$ where $y_i = f(x_i), d_i = f'(x_i)$, Obtain

$$p_i(x) = A_i + B_i(x-x_i) + C_i(x-x_i)^2 + D_i(x-x_{i+1})(x-x_i)^2$$

where last part is for convenience of evaluating constants.
$p_i(x_i) = y_i, p_i'(x_i) = d_i, p_i(x_{i+1}) = y_{i+1}, p_i'(x_{i+1}) = d_{i+1}$

Theorem 19. Suppose f is 4-times differentiable on $[x_i, x_{i+1}]$, p_i is a cubic polynomial such that above satisfies. Then $\forall x \in [x_i, x_{i+1}]$

$$|f(x) - p_i(x)| \leq \max |f^{(4)}(\xi)| \frac{(x_{i+1}-x_i)^4}{384}$$

similar to piecewise linear but now exponent is 4. Then higher accuracy.

5.1.3 Spline interpolation

Required p_i'' be continuous at the knots. Solve for d_i's as part of the algorithm.

$$p_1''(x_2) = p_2''(x_2)$$

$$\vdots$$

$$p_{n-2}''(x_{n-1}) = p_{n-1}''(x_{n-1})$$

1. On each $[x_i, x_{i+1}]$, p is cubic, $i = 1, \cdots, n-1$.
2. $p(x_i) = y_i, i = 1, \cdots, n$.
3. $p'(x_i) = d_i, i = 1, \cdots, n$.

Given $(x_1, y_1), \cdots, (x_n, y_n)$ as above, compute $d_1, \cdots, d_n$, compute cubic Hermite interplant. Determine d_1 to d_n By insisting on second derivative continuity at knots $(x_1, \cdots, x_n)$ satisfies above three bullets and in addition: $p''(x_i)$ is well-defined, $i = 2, \cdots, n-1$ that is p'' is continues over $[x_1, x_n]$. $d_1, \cdots, d_n$ in general cannot be prescribed because this would over-constrain the problem. They must be determining by the interpolation. Must write a system of equations for $d_1, \cdots, d_n$ To satisfy the constraints

$$p_1''(x_2) = p_2''(x_2)$$

$$\vdots$$

$$p_{n-2}''(x_{n-1}) = p_{n-1}''(x_{n-1})$$

where $p_i, i = 1, \cdots, n-1$, is cubic function on $[x_i, x_{i+1}]$. perform some simplification, obtain $n-2$ linear equations that $d_1, \cdots, d_n$ must satisfy format of these equations turns out to be as follows

$$(\Delta x_2)d_1 + 2(\Delta x_1 + \Delta x_2)d_2 + (\Delta x_1)d_3 = \text{RHS involving } x_i, y_i$$

$$(\Delta x_3)d_2 + 2(\Delta x_2 + \Delta x_3)d_3 + (\Delta x_2)dy = \text{another RHS involving } x_i, y_i$$

$n-2$ such equation. Once $d_i's$ determined (solve linear equations). Apply cubic hermit interpolation on each interval. Cubic spline still not uniquely determined. Two equations still needed for d_i's (n unknowns, $n-2$ equations so far). Obtain two more equations by specifying two end conditions. Three common end conditions

1. Complete spline: user specifies d_L, d_R such that

$$p_1'(x_1) = d_L, p_{n-1}'(x_n) = d_R$$

2. Natural spline: impuse constraints

$$p_1''(x_1) = 0$$

$$p_{n-1}''(x_n) = 0$$

3. Not-A-Knot spline: Requires $n \geq 4$. Impose constraints $p'''(x_2) = p_2'''(x_2), p_{n-2}'''(x_{n-1}) = p_{n-1}'''(x_{n-1}) This means that$ p_1, p_2 are actually the same cubic and p_{n-2}, p_{n-1} are actually the same cubic too. So x_2, x_{n-1} are no linear truly knots. The n equations obtained for $d_1, \cdots, d_n$ in all cases are tridiagonal. Say that $A \in \mathbb{R}^{n \times n}$ is tridiagonal. If $A(i, j) = 0$ whenever $|i - j| > 1$. Fact: This linear system may be solved in $O(n)$ operations.

Another method to obtain $d_1, \cdots, d_n$ called pchip in MATLAB. First compute the slopes of piecewise linear interplant to $(x_1, y_1), \cdots, (x_n, y_n)$, call them $\delta_1, \cdots, \delta_{n-1}$. If slopes and opposite sign at a knot, i.e. at $x_k, \delta_{k-1} \cdots \delta_k < 0$, set $d_k = 0$. If slopes same sign, and intervals same length, set $\frac{1}{d_k} = \frac{1}{2}(\frac{1}{\delta_{k-1}} + \frac{1}{\delta_k})$ More general formula for unique lengths. Another formula to determine d_1 and d_n once $d_1, \cdots, d_n$ determined, apply cubic Hermite. In general, not a spline - no second derivative continuity. pchip interpolation does not overshoot data, i.e. local maxima, minima always occur at original data points. PCHIP reserve the shape but a spline is for smoothing.

6 Fourier Transform

6.1 Review of Complex numbers

Cartesian form. Polar form $z = re^{i\Theta}$. Polar representation is not unique since $re^{i\Theta} = re^{i(\Theta+2\pi)}$ etc. Raising to an integer power $(re^{i\Theta})^m = r^m e^{im\Theta}$. Multiples angle by m.

6.2 nth Roots of Unity

nth roots of unity are numbers w such that $w^n = re^{i\Theta}$, e.g., $n = 5$, $(re^{i\Theta})^5 = 1$. This implies that $r = 1$ (to get the length right). $\Theta = k\frac{2\pi}{n}$ k any integer since $e^{iR2\pi} = 1$. Not unique since

$$1, e^{i2\pi/5}, e^{i4\pi/5}, e^{i6\pi/5}$$

then repeat since $e^{i10\pi/5} = e^{i0\pi/5}$, etc. More generally nth roots of unity are

$$e^{-ik2\pi/n}, k = 1, 2, \cdots$$

There are n distinct values. Note: sign is convention for fast Fourier transform.

6.3 Discrete Fourier Transform (DFT)

Given $y_0, \cdots, y_{n-1}$ complex numbers which are coefficients of polynomial.

$$p(z) = y_0 + y_1 z + \cdots + y_{n-1} z^{n-1}$$

Evaluate at nth roots of unity.

$$p(e^{ik2\pi/n})$$

$$Y_k = \sum_{l=1}^{n-1} y_l (e^{ik2\pi/n})^l = \sum_{l=1}^{n-1} y_l e^{ik2\pi l/n}, k = 0, 1, \cdots, n-1$$

Consider inverse problem: given values $Y_0, \cdots, Y_{n-1}$ of an initially unknown polynomial $p(z)$ of degree $\leq n-1$, evaluated at nth roots of unity $e^0, e^{-i2\pi/n}, \cdots, e^{-i2\pi(n-1)/n}$ Find the polynomial $p(z) = y_0 + \cdots + y_{k-1}z^{n-1} i.e. given pairs (\mathrm{e}^{-i2\pi k/n}, Y_k)$ Find standard form (i.e. coefficients) of interpolating polynomial of degree $n-1$: $(y_0, \cdots, y_{n-1})$.

Solution to this interpolation problem is

$$y_l = \frac{1}{n}\sum_{k=0}^{n-1} Y_k e^{i2\pi kl/n}, l = 0, \cdots, n-1$$

IDFT

Proof. Suffices to show that if we start with polynomial $y_0, \cdots, y_{n-1}$ evaluate polynomial at nth roots of unity to obtain $Y_1, \cdots, Y_{n-1}$ and then find coefficients of interpolating polynomial suggested by the formula above should rcover coefficients of original (recall “background” $f(x)$, here a polynomial, in interpolation theory). polynomials, i.e. $\hat{y}_l = y_l$. By IDFT,

$$\begin{aligned} y_l &= \frac{1}{n}\sum_{k=0}^{n-1} Y_k e^{i2\pi kl/n}, l = 0, \cdots, n-1 \\ &= \frac{1}{n}\sum_{k=0}^{n-1}\sum_{j=0}^{n-1} y_j e^{-i} \end{aligned}$$

where Y_k was evaluated by DFT.

Thus it becomes

$$\frac{1}{n}\sum_{j=0}^{n-1} y_j \sum_{k=0}^{n-1} e^{i2\pi(k-j)}$$

inner sum is a geometric series of the form

$$\beta^0 + \beta^1 + \cdots + \beta^{n-1}$$

with $\beta = e^{i2\pi(l-j)/n}$

$$\sum_{k=0}^{n-1} \beta^k = \frac{\beta^k - 1}{\beta - 1}$$

Provided $\beta \neq 1$.

$$\sum_{k=0}^{n-1} Y_k e^{i2\pi kl/n} = \frac{e^{i2\pi(l-j)^n/n} - 1}{e^{i2\pi(l-j)/n} - 1} = 0$$

since

$$e^{i2\pi(l-j)n/n} = 1$$

, $l-j$ integer, except if $\beta = e^{i2\pi(l-j)/n} = 1$, i.e., $\frac{l-j}{n}$ is an integer, i.e., $l-j$ divisible by n, i.e., $l = j$, in which case, each term of the su is 1, so sum is n, i.e.,

$$\hat{y}_l = \frac{1}{n}\sum_{j=0}^{n-1} y_j \begin{cases} 0 & j \neq l \\ n & j = l \end{cases} = \frac{1}{n} y_l n = y_l$$

$\square$