Linguagens de Programação
Desenvolvimento de um programa

Raul Brás

13 de Abril de 2020

Objectivos

Objectivos

Criar um programa para efectuar cálculos comuns em Álgebra Linear

- Operações matriciais
  - soma, subtração, multiplicação
  - transposta, determinante, inversa
- Resolução de sistemas
- Tratamento de erros
  - Dimensões incompatíveis
  - Inversão de matrizes singulares

Algoritmos

Algoritmos

Existem vários algoritmos possíveis para algumas das funcionalidades que queremos implementar.
Vamos usar os que estudaram no MAEG. Nos casos em que estudaram vários usaremos o mais eficiente.
Por eficiência entendemos aqui o algoritmo que executa o menor número de operações.

- soma: $C_{m\times n} = A_{m\times n} + B_{m\times n}$
  $c_{ij} = a_{ij} + b_{ij}, \quad i = 1, \cdots, m,\ j = 1, \ldots, n$
- subtração: $C_{m\times n} = A_{m\times n} - B_{m\times n}$
  $c_{ij} = a_{ij} - b_{ij}, \quad i = 1, \cdots, m,\ j = 1, \ldots, n$
- produto: $C_{m\times n} = A_{m\times p} * B_{p\times n}$
  $c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}, \quad i = 1, \cdots, m,\ j = 1, \ldots, n$

Análise do produto

pseudo código (matrizes $n \times n$):

a) Para $i = 1$ até n
b) Para $j = 1$ até n
c) $c_{ij} \leftarrow 0$
d) Para $k = 1$ até n
e) $c_{ij} \leftarrow c_{ij} + a_{ik} b_{kj}$

a) 1 afectação, n incrementos e n comparações
b) n afectações, n^2 incrementos e n^2 comparações
c) n^2 afectações
d) n^2 afectações, n^3 incrementos e n^3 comparações
e) n^3 afectações, n^3 somas e n^3 produtos

$$(1+2n)+(n+2n^2)+(n^2)+(n^2+2n^3)+(3n^3) \;=\; 5n^3+4n^2+3n+1$$

Este algorimo tem complexidade $\mathcal{O}\left(n^3\right)$

desprezamos constantes e termos de menor grau

Determinante (1)

Pela definição:

$$|A| = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^{n} a_{i\sigma_i}$$

onde:

- S_n é o conjunto de todas as permutações dos índices $1, 2, \cdots, n$.
- σ é uma permutação dos índices
- $\text{sgn}(\sigma)$ é o sinal da permutação: 1 se o número de inversões for par ou -1 se for ímpar.

Como o número total de permutações de n elementos é $n!$ temos $n!$ somas de n produtos.

A complexidade do algorimo é $\mathcal{O}(n!)$, o que é muito mau!

Determinante (2)

Usaremos o método de eliminação de Gauss (condensação).

```
para i=1 até n:
   //Procurar o i-ésimo pivot (máx. abs da coluna):
   indice_max = argmax (k=i ... n, abs(A[k][i]))
   se (A[indice_max][i] == 0) erro "A matriz é singular!"
   se (i != indice_max)
      trocar linhas(i, indice_max)
      incrementar número de trocas
   //Para todas as linhas abaixo do pivot:
   para j=i+1 até n:
     f = A[j][i] / A[i][i]
     //Para o resto dos elementos da linha corrente:
     para k=i+1 até n:
       A[j][k] = A[j][k] - A[i][k] * f
     //Colocar zero nos elementos abaixo do pivot:
     A[j][i] = 0
```

$det = (-1)^s \prod_{i=1}^{n} a_{ii}$, onde s é o número de trocas

Determinante (3)

Contando as instruções aritméticas de vírgula flutuante tem-se:

	Divisões	Mult. e Sub.
$i = 1$	$n - 1$	$(n-1)^2$
$i = 2$	$n - 2$	$(n-2)^2$
$i = 3$	$n - 3$	$(n-3)^2$
$\vdots$	$\vdots$	$\vdots$
$i = n - 2$	2	2^2
$i = n - 1$	1	1
Totais	$\sum_{i=1}^{n-1} n - i$	$\sum_{i=1}^{n-1} (n-i)^2$

$$\sum_{i=1}^{n-1}(n-i) + 2\sum_{i=1}^{n-1}(n-i)^2 = \frac{1}{6}\left(4n^3 - 3n^2 - n\right) \Longrightarrow \mathcal{O}(n^3)$$

Inversa e sistemas

Utilizaremos o algoritmo de condensação aplicado à matriz aumentada com a matriz identidade ou com o segundo membro do sistema.

O algoritmo é semelhante ao anterior.

Nota: existem algoritmos mais eficientes, mas ainda não os estudaram.

Implementação

Implementação tradiconal

- Começaremos por usar apenas funções e iremos melhorando a implementação ao longo da aula, introduzindo classes.
- Criamos um sinónimo para vector<vector<double>>:

```
typedef vector<vector<double>> Matriz
```

- Criamos uma função para criar matrizes $m \times n$:

```
Matriz criaMat(size_t m, size_t n)
{
  if(m==0 || n==0)
    throw errMat{"criaMat: dim. inválidas"};
  //Matriz x(m,vector<double>(n));
  Matriz x;
  x.resize(m);
  for(size_t i=0;i<m;++i) x[i].resize(n);
  return x;
}
```

- Funções para ler e escrever matrizes

```
void leMat(Matriz& x)
{
  if(!x.size() || !x[0].size())
    throw errMat{"leMat: dimensões inválidas"};
  for(size_t i=0;i<x.size();++i)
    for(size_t j=0;j<x[0].size();++j) cin >> x[i][j];
}
void printMat(const Matriz& x)
{
  cout.precision(2);
  for(size_t i=0;i<x.size();++i) {
    for(size_t j=0;j<x[i].size();++j)
      cout << fixed << setw(8) << x[i][j] << ' ';
    cout << '\n';
  }
}
```

- Soma, subtração

```
Matriz soma(const Matriz& A, const Matriz& B)
{
  size_t m=A.size(),n=A[0].size();
  if(m==0 || n==0) throw errMat{"soma: dimensões inválidas"};
  if(m!=B.size() || n!=B[0].size())
    throw errMat{"soma: dimensões incompatíveis"};
  Matriz C=criaMat(m,n);
  for(size_t i=0;i<m;++i)
    for(size_t j=0;j<n;++j) C[i][j]=A[i][j]+B[i][j];
  return C;
}

Matriz subtracao(const Matriz& A, const Matriz& B)
{
  size_t m=A.size(),n=A[0].size();
  if(m==0 || n==0) throw errMat{"subtracao: dimensões inválidas"};
  if(m!=B.size() || n!=B[0].size())
    throw errMat{"subtracao: dimensões incompatíveis"};
  Matriz C=criaMat(m,n);
  for(size_t i=0;i<m;++i)
    for(size_t j=0;j<n;++j) C[i][j]=A[i][j]-B[i][j];
  return C;
}
```

- Produto

```
Matriz produto(const Matriz& A, const Matriz& B)
{
  size_t m=A.size(),p=A[0].size(),n=B[0].size();
  if(m==0 || n==0 || p==0 || B.size()==0)
    throw errMat{"produto: dimensões inválidas"};
  if(p!=B.size()) throw errMat{"produto: dimensões incompatíveis"};
  Matriz C=criaMat(m,n);
  //Trocamos a ordem dos ciclos para optimizar o acesso à cache
  for(size_t i=0;i<m;++i)
    for(size_t k=0;k<p;++k) {
      double aux=A[i][k];
      for(size_t j=0;j<n;++j) C[i][j]+=aux*B[k][j];
    }
  return C;
}
```

- Transposta

```
Matriz transp(const Matriz& A)
{
  size_t m=A.size(),n=A[0].size();
  if(m==0 || n==0) throw errMat{"produto: dimensões inválidas"};
  Matriz T=criaMat(n,m);
  for(size_t i=0;i<m;++i)
    for(size_t j=0;j<n;++j) T[j][i]=A[i][j];
  return T;
}
```

Determinante

```
bool isZero(double x) { return abs(x)<1e-14;}
double det(const Matriz& A)
{
  int s=1; //sinal do nº de trocas
  size_t m=A.size(),n=A[0].size();
  if(m==0 || n==0) throw errMat{"det: dimensões inválidas"};
  if(m!=n)  throw errMat{"det: a matriz não é quadrada"};
  Matriz x=A;
  for(size_t i=0;i<n;++i) {
    //o pivot é o maior elemento, em valor absoluto, da coluna
    size_t idx=i;
    double max=abs(x[i][i]);
    for(size_t k=i+1;k<n;++k) {
      if(abs(x[k][i])> max) {max=abs(x[k][i]); idx=k;}
    }
    if(isZero(max)) return 0.0; //a matriz é singular
    if(idx!=i) { swap(x[i],x[idx]); s=-s; }
    for(size_t k=i+1;k<n;++k) {
      double p=x[k][i]/x[i][i];
      for(size_t j=i+1;j<n;++j) x[k][j]-=x[i][j]*p;
      x[k][i]=0.0;
    }
  }
  double d=s; //1 ou -1
  for(size_t i=0;i<n;++i) d*=x[i][i];
  return d;
}
```

Inversa

```
Matriz inverte(Matriz x)
{
  size_t m=x.size(),n=x[0].size();
  if(m==0 || n==0) throw errMat{"inversa: dimensões inválidas"};
  if(m!=n)  throw errMat{"inversa: a matriz não é quadrada"};
  Matriz inv=criaMat(n,n); //É inicializada com 0
  for(size_t i=0;i<n;++i) inv[i][i]=1;
  for(size_t i=0;i<n;++i) {
    size_t idx=i;
    double max=abs(x[i][i]);
    for(size_t k=i+1;k<n;++k)
      if(abs(x[k][i])> max) {max=abs(x[k][i]); idx=k;}
    if(isZero(max)) throw errMat{"inversa: a matriz é singular\n"};
    if(idx!=i) { swap(x[i],x[idx]); swap(inv[i],inv[idx]); }
    for(size_t k=i+1;k<n;++k) {
      double p=x[k][i]/x[i][i];
      for(size_t j=i+1;j<n;++j) x[k][j]-=x[i][j]*p;
      x[k][i]=0;
      for(size_t j=0;j<n;++j) inv[k][j]-=inv[i][j]*p;
    }
  }
  //Fim da condensação. Agora temos que condensar de baixo para cima
  //Começamos por colocar 1 na diagonal principal
  for(size_t i=0;i<n;++i) {
    double d=x[i][i];
    for(size_t j=0;j<n;++j) {x[i][j]/=d; inv[i][j]/=d;}
  }
  //agora condensamos
  for(int i=n-1;i>=1;--i) {
    for(int k=i-1;k>=0;--k) {
      double p=x[k][i]/x[i][i];
      for(size_t j=i+1;j<n;++j) x[k][j]-=x[i][j]*p;
      x[k][i]=0;
      for(size_t j=0;j<n;++j) inv[k][j]-=inv[i][j]*p;
    }
  }
  return inv;
}
```

Implementação com classes

Implementação com classes

- Problemas com a implemetação tradicional
  - O acesso aos dados das matrizes é livre. É fácil cometer erros e alterar os dados ou as dimensões de uma matriz
  - Não nos podemos esquecer de chamar criaMat. Gostaríamos que fosse automático
  - Não podemos escrever por ex. A=B*C+D.inversa();

Implementação com classes

- classe Matriz
  - armazena os dados
  - executa as operações matriciais
- class LinAlgebra
  - resolve sistemas
  - etc.
- class errMat
  - tratamento dos erros

Classe Matriz

```
class Matriz {
 private:
   std::vector<std::vector<double>> mat;
   const size_t m,n;
 public:
   Matriz(size_t mm, size_t nn);

   double det() const;
   Matriz inverte() const;
   Matriz transp() const;
   std::vector<double>& operator[](size_t i)
       {return mat[i];}
   const std::vector<double>& operator[](size_t i) const
       {return mat[i];}

   size_t lines() const {return m;}
   size_t cols() const {return n;}
 private:
   bool isZero(double x) const {return std::abs(x)<1e-14;}
};
```

Funções auxiliares

Podem também ser membros da classe, com excepção dos operadores para efectuar o produto por um escalar.

```
std::istream& operator>>(std::istream&,Matriz&);
std::ostream& operator<<(std::ostream&,const Matriz&);
Matriz operator+(const Matriz&, const Matriz&);
Matriz operator-(const Matriz&, const Matriz&);
Matriz operator-(const Matriz&);
Matriz operator*(const Matriz&, const Matriz&);
Matriz operator*(double, const Matriz&);
Matriz operator*(const Matriz&, double);
bool operator==(const Matriz&, const Matriz&);
bool operator!=(const Matriz&, const Matriz&);
```

Construtor

```
Matriz::Matriz(size_t mm, size_t nn) : m{mm},n{nn}
{
  if(m==0 || n==0)
    throw errMat{"Matriz::Matriz - dimensões inválidas"};
  mat.resize(m);
  for(size_t i=0;i<m;++i) mat[i].resize(n,0.0);
}
```

Operadores de Input e Output

```
istream& operator>>(istream& is, Matriz& x)
{
  for(size_t i=0;i<x.lines();++i)
    for(size_t j=0;j<x.cols();++j) is >> x[i][j];
  return is;
}

ostream& operator<<(ostream& os, const Matriz& x)
{
  os.precision(2);
  for(size_t i=0;i<x.lines();++i) {
    for(size_t j=0;j<x.cols();++j)
      os << fixed << setw(8) << x[i][j] << ' ';
    os << '\n';
  }
  return os;
}
```