Numerical Methods for CSE

David Zollikofer

Stuff I tend to forget

- `MatrixXd:Ones()` is not `MatrixXd::Identity()`
- Frobenius Norm ist die normale $\sqrt{\cdot}$ Norm mit dem Befehl `.norm()`.

C++ Objects - Initializer Lists

```
class Point {
  private:
    int x;
    int y;
  public:
    Point(int i = 0, int j = 0):x(i), y(j) {}
      /*  The above use of Initializer list is optional as the
      constructor can also be written as:
      Point(int i = 0, int j = 0) {
      x = i;
      y = j;
    }
  */
};
```

Generics

Functions and Generics

```
template <typename T>
T mx(T x, T y){
  return (x > y) ? x : y;
}
int main() {
  cout << mx<int>(3, 7) << mx<double>(3.0, 7.0) << mx<char>('g', 'e');
  return 0;
}
```

Classes and Generics

```
template <typename T, typename U>
class A {
  T x;
  U y;
  public:
    A() { cout << "constructor";}
};
```

```
int main() {
  A<char, char> a;
  A<int, double> b;
}
```

Lambda Functions

Lambda Functions & Applying Coefficient wise

```
auto phi1 = [](int t) -> double {return 1/t;};
auto phi2 = [](int t){return 1/(t*t);};
b = b.unaryExpr(phi1);
```

```
#include <functional>
int main(){
  std::function<double(int)>  f = [](int l){
    return l + 3.14;
  };
  std::cout << f(2);
}
```

Using std::vector<...>

Turn std::vector<...> to VectorXd

```
std: :vector<double> t {12};
Map<VectorXd> te (t.data(),t.size());
```

General Usage

```
#include <vector>
std::vector<int> M {10,20,40,80,160,320,640,1280};
// Create a vector of size n with
// all values as 10.
vector<int> vect(n, 10);
vector<int> g1;
g1.push_back(123);
g1.size();
g1.resize(4); // does change size
g1.reserve(100); // doesn't change size
cout << g1[2];
```

Using std::pair<...,...>

```
#include <utility>
std::pair<int, char> pr1;
pr2 = make_pair("some text",4.56);
pr1.first = 100;
pr2.second = 'G' ;
std::cout << pr1.first << " " << pr1.second ;
```

Mathplotlib Plotting

```
// plotting results
plt::figure();
plt::plot(t, y, "+", {{"label", "approx IVP"}});
// exact solution
VectorXd x = Vector::LinSpaced(100,0.0,T);
VectorXd exact = x.array().tan();
plt::plot(x, exact, {{"label", "tangent"}}  );
plt::legend();

plt::title("Approximate vs exact solution");
plt::xlabel("t");
plt::ylabel("y");
plt::grid("True");

plt::savefig("./cx_out/fileName.png");
```

Scientific Notation in C++

The number `42030` can also be seen as `4.203e4` since we shift the decimal point 4 places to the left. Analogous with shifting to the right.

Constructing Matrices and Vectors

```
// same data different shape
Map<Matrix<TYPE, Dynamic , Dynamic>> newMat(M.data(),newRows,newCols);
// (Deep) copy data of M
MatrixXd xMat = MatrixXd::Map(M.data(), n, n);
```

Sizes, Rows, Cols, Resize

```
A.size() // total number of elements in matrix
A.rows() // number of rows
A.cols() // number of cols
A.resize(10,100) // resized to 10 rows and 100 columns
```

Filling matrices

```
A << R, v // can also be done using blocks.
u .transpose( ) ,  0;
B << A, A, A;      // B is three horizontally stacked A's.
A.fill(10);        // Fill A with all 10's.
```

```
MatrixXd::Identity(rows,cols)
C.setIdentity(rows,cols)
MatrixXd::Zero(rows,cols)
C.setZero(rows,cols)
MatrixXd::Ones(rows,cols)
C.setOnes(rows,cols)
MatrixXd::Random(rows,cols)  // uniform random numbers in (-1,1).
C.setRandom(rows,cols)
VectorXd::LinSpaced(size,low,high)        // includes lower and higher
  bound -> if size=1 only higher.
v.setLinSpaced(size,low,high)
VectorXi::LinSpaced(((hi-low)/step)+1, low,low+step*(size-1))
```

Reshaping using Map

```
// same data different shape
Map<Matrix<TYPE, Dynamic , Dynamic>> newMat(M.data(),newRows,newCols);
// (Deep) copy data of M
MatrixXd xMat = MatrixXd::Map(M.data(), n, n);
```

Solving Equations

```
// Solve Ax = b. Result stored in x
x = A.ldlt().solve(b));  // A sym. p.s.d.     #include <Eigen/Cholesky>
x = A.llt().solve(b));  // A sym. p.d.       #include <Eigen/Cholesky>
x = A.lu().solve(b));  // Stable and fast. #include <Eigen/LU>
x = A.qr().solve(b));  // No pivoting.     #include <Eigen/QR>
x = A.svd().solve(b));  // Stable, slowest. #include <Eigen/SVD>
// .ldlt() -> .matrixL() and .matrixD()
// .llt()  -> .matrixL()
// .lu()   -> .matrixL() and .matrixU()
// .qr()   -> .matrixQ() and .matrixR()
// .svd()  -> .matrixU(), .singularValues(), and .matrixV()
```

Block

Block of size (p,q), starting at (i,j) is `A.block(i,j,p,q);`

Triangular Views We have the following options `Upper`, `Lower`, `StrictlyUpper`, `StrictlyLower`, `UnitUpper`, `UnitLower`

```
MatrixXd m2 = m.triangularView<Xxx>();
```

Diagonal

```
A = MatrixXd::Constant(n,n,0); // initialization is necessary
vec1 = mat1.diagonal();
mat1.diagonal() = vec1;
```

Reductions

```
R.minCoeff()
R.maxCoeff()
s = R.minCoeff(&r, &c)
R.sum()
R.colwise().sum()
R.prod()
R.rowwise().prod()
```

```
R.trace()
(a > 2).all() // if all a_i > 2 true
R.rowwise().all()
(a > 2).any()    // if any a_i > 2 true
(a > 2).count() // count a_i > 2
R.colwise().any()
x.norm()
x.squaredNorm()
x.dot(y)
x.cross(y)     //Requires #include <Eigen/Geometry>
```

Kronecker Product

```
  C = kroneckerProduct(A,B).eval();
```

Sparse Matrices

Defining sparse matrices

```
#include <Eigen/Sparse>
SparseMatrix<double> mat(rows,cols);
std::vector<Triplet<double>> tripletVec;
tripletVec.reserve(n);
for(int i = 0; i < n; i++){
  double val = 2;
  Triplet<double> newTriplet(row,col,val);
  tripletVec.push_back(newTriplet);
}
X.setFromTriplets(tripletVec.begin(), tripletVec.end());
X.makeCompressed();
```

Solving Linear Equations with Sparse Matrices

```
Eigen::SparseMatrix<double> A(n,n);
A.reserve(3*n);
for(;;){
  A.insert(i,i-1) = 100;
}
A.makeCompressed(); // VERY IMPORTANT
Eigen::SparseLU<Eigen::SparseMatrix<double>> Ax_lu(A)
newX = Ax_lu .solve(b);
```

Using sparse matrixes

```
  //now we loop over every non zero entry of M (for the left hand size
   of the kronecker product)
  // loop over every column
  for(int i = 0; i < n;i++){
    // we loop as long as there are still elements in the column
    // we get the pointer to where the column starts and loop for as
      long as elements still are in this column
    for(int index = colPointer[i]; index < colPointer[i+1] ; index++){
      // we are now at the nonzero element in row "i" and column "j" and
         value "outerValue".
      int outerRow = rowPointer[index];
      int outerColumn = i;
      double outerValue = values[index];

      // now we need to loop over all of M again to add the block to the
        kronecker matrix

      for(int innerI = 0; innerI < n;innerI++){
        // we loop as long as there are still elements in the row
        for(int innerIndex = colPointer[innerI]; innerIndex < colPointer
         [innerI+1] ; innerIndex++){

          int innerRow = rowPointer[innerIndex];
          int innerColumn = innerI;
          double innerValue = values[innerIndex];


          Triplet<double> toAdd(n*outerRow+innerRow,n*outerColumn+
           innerColumn,outerValue*innerValue);
          bVec.push_back(toAdd);
        }
      }
    }
  }
```

QR Decompositions

Householder Reflection

```
function [Q,R] = qr_householder(A)
%          Spiegelungen
%         R ... obere (upper) Dreiecks-Matrix
% Speicherplatz fuer Q & R
[m,n] = size(A);
Q = zeros(m,m);
R = zeros(m,n);
% Speicherplatz fuer die Householder-Spiegelungs Vektoren
U = zeros(m,n);
% QR-Zerlegung mit Householder-TransformationenBitte wenden!
R = A;
for j=1:n
% wir extrahieren die zu eliminierende spalte
w = R(j:m,j);
% falls m < n ist, breche bei hier ab wenn j > m
if ( j > m )
break
end
% waehle Spiegelung mit ?kleinster? Wirkung
if ( w(1) >= 0 )
w(1) = w(1) - norm(w);
else
```

```
w(1) = w(1) + norm(w);
end
% normiere Spiegelungs Vektor
u = w/(norm(w) + eps);
% speichere normierten Spieglungs Vektor in U% wende die Householder-
 Spiegelung auf A(j:m,j:n) an
U(j:m,j) = u;
for i=j:n
R(j:m,i) = R(j:m,i) - 2.*u*(u'*R(j:m,i));
end
end% bauen nun Q? auf
Q = eye(m) - 2.*U(:,1)*U(:,1)';
for i=2:n
H = eye(m) - 2.*U(:,i)*U(:,i)';
Q = H*Q;
end
Q = Q';
```

Givens Rotation Grundlegende Idee. Wir killen alle Einträge unterhalb der Diagonale in A und bauen uns R draus.

$$\underbrace{\mathbb{1}G_1^{-1}\dots G_k^{-1}}_{=:Q}\underbrace{G_k\dots G_1A}_{=:R}$$

```
function [Q,R] = qrgivens(A)
  [m,n] = size(A);
  Q = eye(m);
  R = A;
  for j = 1:n
    for i = m:-1:(j+1)
      G = eye(m);
      [c,s] = givensrotation( R(i-1,j),R(i,j) );
      G([i-1, i],[i-1, i]) = [c, s; -s, c];
      R = G*R;
      Q = Q*G';
    end
  end
end
function [c,s] = givensrotation(a,b)
  if b == 0
    c = 1;
    s = 0;
  else
    if abs(b) > abs(a)
      r = a / b;
      s = 1 / sqrt(1 + r^2);
      c = s*r;
    else
      r = b / a;
      c = 1 / sqrt(1 + r^2);
      s = c*r;
    end
  end
```

```
end
```

Householder QR taken from code 3.3.3.29

```
#include<Eigen/QR>
Eigen::HouseholderQR<MatrixXd> qr(A);
MatrixXd Q = qr.householderQ();
MatrixXd R = qr.matrixQR().template triangularView <Eigen::Upper>();
```

Least Squares

Introduction

We are interested in solving a problem $\mathbf{Ax} = \mathbf{b}$, where the system is overdeterminded, there might not exist a solution. We are aiming for $x \in \text{argmin}_{y \in \mathbb{K}^n} \|Ay - b\|_2^2$.

Solve if A has full rank:

We have two options:

- **Normal Equation** $A^T Ax = A^T b$

```
VectorXd x = (A.transpose()*A).llt().solve(A.transpose()*b);
```

- **QR decomposition** $\|Ax - b\|_2 = \|Q(Rx - Q^H b)\|_2 = \|Rx - Q^H b\|_2$

```
VectorXd x = A.householderQr().solve(b);
```

Not full rank A:

We get infinitely many solutions if **A** doesn't have full rank. In this case we are interested in the

We define the **generalized solution** $x^+ := \text{argmin}\{\|x\|_2 : x \in \text{lsq}(A, b)\}$. Use Theorem 3.1.3.6 (Formula for generalized solution).

For this we use the Singular Value Decomposition as seen in code 3.4.3.7 (Computing generalized solution of $Ax = b$ via SVD)

```
JacobiSVD<MatrixXd> svd(A,ComputeThinU | ComputeThinV);
VectorXd solution = svd.solve(b);
```

Constrained Least Squares (3.6) Assume we want to minimize $\|Ax - b\|_2$ under the constraint $Cx = d$. We use Lagrangian multipliers and get the *augmented normal equations*:

$$\begin{bmatrix} \mathbf{A}^T\mathbf{A} & \mathbf{C}^T \\ \mathbf{C} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{m} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^T\mathbf{b} \\ \mathbf{d} \end{bmatrix}$$

Alternatively we can also use the extended normal equations and get

$$\begin{bmatrix} -\mathbf{I} & \mathbf{A} & 0 \\ \mathbf{A}^T & 0 & \mathbf{C}^T \\ 0 & \mathbf{C} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{x} \\ \mathbf{m} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ 0 \\ \mathbf{d} \end{bmatrix}$$

Gaussian Block Elimination

Example:

$$\begin{bmatrix} \mathbb{1} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x} \\ \mathbf{m} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{g} \end{bmatrix}$$
$$\Rightarrow \mathbf{1} \cdot \mathbf{x} + \mathbf{C}^T \cdot \mathbf{m} = \mathbf{0} \Rightarrow -\mathbf{C}^T\mathbf{m} = \mathbf{x}$$
$$\Rightarrow \mathbf{Cx} = \mathbf{g}$$
$$\Rightarrow -\mathbf{C} \cdot (\mathbf{C}^T\mathbf{m}) = \mathbf{g}$$

Data Interpolation and Fitting (Chapter 5)

There are three main approaches:

- Using Vandermonde Matrix $\rightarrow$ never do this. Very badly conditioned matrix.
- **Lagrange** Good choice, use Barycentric if interested in coeffs, if only need one eval use Aitken-Neville.
- **Newton** If we dynamically add new data (update friendly) (5.2.3.23)

Extraplation to 0: Works really well if $\phi(t) = \phi(-t)$, we use Aitken-Neville on extrapolation point.

Function Approximation (Chapter 6)

Approximation is the combination of an interpolation scheme and sampling. We get to choose where to evaluate the function f.

Jackson's Theorem (6.1.1.11) If $f \in C^r([-1,1])$ ($r \in \mathbb{N}$ times continuously differentiable), then for any polynomial of degree $n \geqslant r$

$$\inf_{p \in \mathcal{P}} \|f - p\|_{L^\infty[(-1,1)]} \leqslant (1 + \frac{\pi^2}{2})^r \frac{(n-r)!}{n!} \|f^{(r)}\|_{L^\infty[(-1,1)]}$$

We can also apply Jackson's Theorem to any interval using Lemma 6.1.1.20

Generally use Chebychev interpolation, (better at controlling phenomena such as Runge's problem).

Convergence

- Algebraic Convergence: data on line in log-log plot, $\exists p > 0 : T(n) \leqslant n^{-p}$
- Exponential Convergence: data on line in lin-log plot , $\exists 0 < q < 1 : T(n) \leqslant q^n$

Quadrature (Chapter 7)

Basic Idea. We would like to calculate $\int_a^b f(x)dx = \sum_{j=1}^n w_j \cdot f(c_j)$, where w_j are weights and c_j are nodes.

A *Quadrature Rule* usually specifies weights and nodes on the interval $[-1, 1]$, we then transform them accordingly to $[a, b]$

$$w_i \mapsto \frac{(b-a)}{2} \cdot w_i$$
$$c_i \mapsto \frac{1}{2}(1 - c_i)a + \frac{1}{2}(1 + c_i)b$$

The **Quadrature Error** is given by $E_n(f) = \left|\int_a^b f(t)dt - Q_n(f)\right|$

Types of quadrature error

- ALGEBRAIC CONVERGENCE $E(n) = \mathcal{O}(n^{-p}), p > 0$
- EXPONENTIAL CONVERGENCE $E(n) = \mathcal{O}(q^n), 0 \leqslant q < 1$, here use $\log$ on the error and then do linear fitting.

Order of quadrature rules

Definition 7.3.0.1. Order of a quadrature rule

The order of quadrature rule $Q_n : C^0([a,b]) \to \mathbb{R}$ is defined as

$$\text{order}(Q_n) := \max\{m \in \mathbb{N}_0 : \; Q_n(p) = \int_a^b p(t)\,dt \quad \forall p \in \mathcal{P}_m\} + 1\,, \tag{7.3.0.2}$$

that is, as the maximal degree $+1$ of polynomials for which the quadrature rule is guaranteed to be exact.

Grundsätzlich: Newton-Cotes haben äquidistante Intervalle. Auf diesen Integrieren wir Lagrange Polynome, dies sind dann unsere Gewichte.

n		Order
1	midpoint rule	2
2	trapezoidal rule	2
3	simpson rule	4
4	$\frac{3}{8}$-rule	4
5	Milne rule	6
n	clenshaw curtis (from chebyshev, has all positive weights)	n
n	gauss legendre	2n

Composite Quadrature

Chop the integral into smaller integrals and integrate them seperately. Has algebraic convergence.

Order of Convergence

Drawing on best approximation estimates from Section 6.1.1 and Rem. 6.1.3.22, we immediately get results about the asymptotic decay of the quadrature error for n-point Gauss-Legendre and Clenshaw-Curtis quadrature as $n \to \infty$:

$$\begin{aligned} f \in C^r([a,b]) \quad &\Rightarrow \quad E_n(f) \to 0 \text{ algebraically with rate } r, \\ f \in C^\infty([a,b]) \text{ “analytically extensible”} \quad &\Rightarrow \quad E_n(f) \to 0 \text{ exponentially}, \end{aligned}$$

as $n \to \infty$, see Def. 6.1.2.7 for type of convergence.

Iterative Meth for nonlin. Sys. of Equations (Chapter 8)

Nonlinear least squares

Basic idea (can be disgusting): Use Gauss to find where $\text{grad}(\phi) = 0$, this will be a minimal solution.

Better Idea: Use Gauss-Newton method. We linearize $\tilde{f}(x) = f(x^0) + \nabla f(x^0)^T(x - x^0)$ and then minimize $\min_{x\in\mathbb{R}^n}\left\{\frac{1}{2}\|\tilde{f}(x)\|^2 = \frac{1}{2}\|J(x-x^0) + f(x^0)\|^2\right\}$. Now we set $\nabla\tilde{f}(x) = J^T\left(J(x-x^0) + f(x^0)\right) = 0$
See procedure in last subsection of chapter 8.

Differential Equations

Transforming to autonomous 1st order ODE

Assume we have $y'' = -g\sin(y)$, we define $u_1 = y$ and $u_2 = y'$. This yields:

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} y \\ y' \end{pmatrix} \qquad f(t,\mathbf{u}) = \begin{pmatrix} u_2 \\ -g\sin(u_1) \end{pmatrix} = \begin{pmatrix} u_1' \\ u_2' \end{pmatrix} = \mathbf{u}'$$

We can also get rid of a dependence on t (make autonomous): Assume we have $y' = -y + \cos(t)$. We define $v_1 = y$ and $v_2 = t$, this yields:

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} y \\ t \end{pmatrix} \qquad f(\mathbf{v}) = \begin{pmatrix} -v_1 + \cos(v_2) \\ 1 \end{pmatrix} = \begin{pmatrix} v_1' \\ v_2' \end{pmatrix} = \mathbf{v}'$$

Runge Kutta Methods

We have the IVP $\dot{\mathbf{y}} = \mathbf{f}(t,\mathbf{y}), \mathbf{y}(t_0) = \mathbf{y_0}$. We can write $\mathbf{y_1}$ as

$$\mathbf{y}(t_1) = \mathbf{y_0} + \int_{t_0}^{t_1} \mathbf{f}(\tau, \mathbf{y}(\tau))\,d\tau$$

giving us

$$\mathbf{y}(t_1) \approx \mathbf{y_0} + h\sum_{i=1}^{s} b_i \mathbf{f}(t_0 + c_i h, \mathbf{y}(t_0 + c_i h))$$

Butcher Scheme We call a single step Runge Kutta method consistent iff.

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array} := \begin{array}{c|cccc} c_1 & 0 & \cdots & \cdots & 0 \\ c_2 & a_{21} & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{s,s_1} & 0 \\ \hline & b_1 & \cdots & \cdots b_{s-1} & b_s \end{array}$$

Find Order of Convergence of Runge-Kutta

```
VectorXd coeffs = polyfit(nArr.array().log(), err.array().log(),1);
conv_rate = -coeffs(0);
```

Stiff Problems

Calculating the stability function:

The stability function $S(z)$ is defined as $y_1 = S(z)y(0)$ for any Runge-Kutta method.

- **Explicit Euler** $y_1 = y_0 + h\cdot f(y_0)$, with model problem $y' = \lambda y$, hence we have $y_1 = y_0 + h\cdot\lambda\cdot y_0 = y_0\,(1+\lambda h)$. Using $z = h\lambda$ we get $S(z) = 1 + z$.
- **Trapezoidal rule** $y_1 = y_0 + h\frac{1}{2}(f(y_0) + f(y_0 + hf(y_0)))$ with $y' = \lambda y$, hence $y_1 = y_0 + h\frac{1}{2}(\lambda y_0 + \lambda y_0 + \lambda^2 h y_0)$ giving us $y_1 = y_0 + h\lambda y_0 + \frac{\lambda^2 h^2 y_0}{2}$ resulting in $y_1 = y_0\left(1 + h\lambda + \frac{\lambda^2 h^2}{2}\right)$ which yields the stability function $S(z) = 1 + z + \frac{z^2}{2}$.

Stability Function via Butcher Table The discrete evolution Ψ^h_λ of an s-stage Runge-Kutta single step method with Butcher scheme $\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array}$, the stability function is given by:

$$S(z) = 1 + z\mathbf{b}^T(\mathbf{I} - z\mathbf{A})^{-1}\mathbf{1}$$

where $\mathbf{1} = [1,\ldots,1]^T$

Region of (absolute) Stability

$$S : \Phi := \{z \in \mathbb{C} : |S(z)| < 1\} \subseteq \mathbb{C}$$

This definition follows from the model problem $y' = \lambda y$ where we only have solutions that don't blow up if $|S(\lambda h)| < 1$.

Stiff Problems A problem is stiff if it imposes much tighter timestep constraints (to keep the accuracy) on explicit single time step methods than the accuracy would usually require.
An IVP for an autonomous ODE $\mathbf{y}' = \mathbf{f}(\mathbf{y})$ will probably be stiff, if, for substantial periods of time:

$$\min\{\operatorname{Re}\lambda : \lambda \in \sigma(D\mathbf{f}(\mathbf{y}(t)))\} \ll 0$$

where $t \to \mathbf{y}(t)$ is the trajectory and σ is the set of eigenvalues. See example 12.2.0.16 for an example.
Furthermore, if $\max\{\operatorname{Re}\lambda : \lambda \in \sigma(D\mathbf{f}(\mathbf{y}(t)))\} \approx 0$, we are unlikely to see blow ups.

Stability

A stability of RK method A Runge-Kutta single step method with stability function S is A-stable if

$$\mathbb{C}^- := \{z \in \mathbb{C} : \operatorname{Re} Z < 0\} \subseteq S_\Phi$$

where S_Φ is the stability region.
Die Idee dahinter: Wenn die Testfunktion nicht explodiert $h\lambda = z \leqslant 0$, dann sollte das Verfahren für alle Zeitschritte nicht explodieren.

Nun gibt es aber das Problem, dass wir langfristig nicht explodieren aber grosse Diskretisierungsfehler haben. Um dies zu vermeiden gibt es:

L stability of RK method If the stability function $S(z)$ satisfies:

$$\operatorname{Re} z < 0 \Rightarrow |S(z)| < 1$$

$$\lim_{\operatorname{Re} z \to -\infty} S(z) = 0$$

Insbesondere gilt L-stabil $\iff$ A-stabil & "$S(-\infty) = 0$"

Implicit Euler is L stable and Implicit midpoint is A stable.

Integrale Ausrechnen

Direkte Integrale

Diese sind vom Typ $\int f(g(x))g'(x)dx = F(g(x))$.

Partielle Integration

Partielle Integration

$$\int f' \cdot g \, dx = f \cdot g - \int f \cdot g' \, dx$$

Beispiele:

Integrale rationaler Funktionen

Partielle Integration

$$\int \frac{p(x)}{q(x)} dx$$

Wenn nun $\deg(p) \geqslant \deg(q)$ dann machen wir eine Polynomdivision $p : q$, sonst mache eine Parzialbruchzerlegung

Substitutionsregel

Substitutionsregel Ist f stetig und g erfüllt:

$$y = g(x) \iff x = g^{-1}(y)$$

Dann gilt:

$$\int_a^b f(g(x))g'(x)dx = \int_{g(a)}^{g(b)} f(y)dy$$

Als Merksatz gilt $dy = g'(x)dx$ respektive $dx = \frac{1}{t}dt$

Matrixinverse

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Funktion	Ableitung	Bemerkung / Regel
x	1	
x^2	$2x$	
x^n	$n \cdot x^{n-1}$	$n \in \mathbb{R}$
$\frac{1}{x} = x^{-1}$	$-\frac{1}{x^2}$	
$\sqrt{x} = x^{\frac{1}{2}}$	$\frac{1}{2\sqrt{x}}$	
$\sqrt[n]{x} = x^{\frac{1}{n}}$	$\frac{x^{\frac{1}{n}-1}}{n}$	$\int x^{1/n} dx = \frac{nx^{1/n+1}}{n+1} + C$
e^x	e^x	
a^x	$\ln(a) \cdot a^x$	
$x^x = e^{x \log(x)}$	$x^x \cdot (\log(x) + 1)$	Kettenregel $e^{x \log(x)}$
$\ln(x)$	$\frac{1}{x}$	
$x \ln(x) - x$	$\ln(x)$	
$\sin(x)$	$\cos(x)$	
$\cos(x)$	$-\sin(x)$	
$\tan(x) = \frac{\sin(x)}{\cos(x)}$	$\frac{1}{\cos^2(x)} = 1 + \tan^2(x)$	
$\cot(x) = \frac{\cos(x)}{\sin(x)}$	- $\frac{1}{\sin^2(x)}$	
$\arcsin(x)$	$\frac{1}{\sqrt{1-x^2}}$	$\arcsin : [-1, 1] \to [-\frac{\pi}{2}, \frac{\pi}{2}]$
$\arccos(x)$	$-\frac{1}{\sqrt{1-x^2}}$	$\arccos : [-1, 1] \to [0, \pi]$
$\arctan(x)$	$\frac{1}{1+x^2}$	$\arctan : (-\infty, \infty) \to (-\frac{\pi}{2}, \frac{\pi}{2})$
$\text{arccot}(x)$	$-\frac{1}{1+x^2}$	$\text{arccot} : (-\infty, \infty) \to (0, \pi)$
$\cosh(x)$	$\sinh(x)$	
$\sinh(x)$	$\cosh(x)$	
$\tanh(x)$	$\frac{1}{\cosh^2(x)}$	
$\text{arsinh}(x)$	$\frac{1}{\sqrt{1+x^2}}$	$\forall x \in R$
$\text{arcos}(x)$	$\frac{1}{\sqrt{x^2-1}}$	$\forall x \in (1, \infty)$
$\text{artanh}(x)$	$\frac{1}{1-x^2}$	$\forall x \in (-1, 1)$
$g(x) \cdot h(x)$	$g(x) \cdot h'(x) + g'(x) \cdot h(x)$	Produktregel
$(g(x))^n$	$n \cdot (g(x))^{n-1} \cdot g'(x)$	Potenzregel
$\frac{g(x)}{h(x)}$	$\frac{g'(x) \cdot h(x) - g(x) \cdot h'(x)}{(h(x))^2}$	Quotientenregel
$h(g(x))$	$h'(g(x)) \cdot g'(x)$	Kettenregel

Multidimensional Derivative using the Definition

Example (Total Differentiability) Let $f(0,0) = 0$ and $f(x,y) = \frac{x^2y^2}{x^2+y^2}$ for $(x,y) \neq (0,0)$. We calculate the Jacobi Matrix at $(0,0)$ and plug it into the definition of differentability:

$$\frac{\partial f}{\partial x}(0,0) = \lim_{h \to 0} \frac{f(0+h,0) - f(0,0)}{h} = \lim_{h \to 0} \frac{\frac{0h^2}{h^2+0} - 0}{h} = 0$$

$$\frac{\partial f}{\partial y}(0,0) = \lim_{h \to 0} \frac{f(0,0+h) - f(0,0)}{h} = \lim_{h \to 0} \frac{\frac{h^2 0}{0+h^2} - 0}{h} = 0$$

Hence we have $J_f(0,0) = [0,0]$. Now we calculate:

$$\begin{aligned}
&\lim_{(x,y) \to (0,0)} \frac{\left(f\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) - f\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}\right) - J_f(0,0) \begin{pmatrix} x-0 \\ y-0 \end{pmatrix}\right)}{\left\| \begin{pmatrix} x-0 \\ y-0 \end{pmatrix} \right\|} \\
&= \lim_{(x,y) \to (0,0)} \frac{\frac{x^2y^2}{x^2+y^2} - 0 - [0,0] \begin{pmatrix} x-0 \\ y-0 \end{pmatrix}}{\sqrt{x^2+y^2}} \\
&= \lim_{(x,y) \to (0,0)} \frac{\frac{x^2y^2}{x^2+y^2}}{\sqrt{x^2+y^2}} \\
&= \lim_{r \to 0} \frac{r^4 \sin^2(\varphi) \cos^2(\varphi)}{r^3} = 0
\end{aligned}$$