## Programm erstellen

1. Algorithmus auswählen
2. Ein- und Ausgabevariabeln festlegen
3. Kommentare schreiben
4. Interne Variabeln festlegen
5. Programm schreiben
6. Testen
7. Evtl. optimieren
8. Numerisches Debuggen
9. Sicherheits-/ Echtzeitanforderungen beachten

## Verfahren numerisches Lösen

- Runge Kutta 4: Lösen von Dgls
- Trapezregel: Numerisches Integrieren
- Newton-Raphson: Nullstellen
- Newton: Optimierung

## Bedingungen Well-Posed Problem

- Existenz einer Lösung
- Eindeutigkeit der Lösung
- Hadamard-Bedingungen: stetige Abhänigkeit von den Eingansdaten

## Kennzeichen von Testmatrizen

1. In der Ordnung skalierbar (z.B. test des Speicherverbrauchs
2. Lösung bekannt
3. Testen numerische Grenzfälle (z.B. Singularitäten)
4. In der Kondition variabel
5. Literatur/Tabellenwerke

## Numerisches Differenzieren

$$f_k^{(n)} = \frac{\sum_{i=-p}^{q} w_{k+i} f_{k+i}}{h^n}$$

| Ableitung | Stützstellen | Polynom grad | fk+3 | $f_{k+2}$ | $f_{k+1}$ | $f_k$ | $f_{k-1}$ |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | | | | 1 | -1 |
| | 2 | 1 | | | 1 | -1 | |
| | 3 | 2 | | | 1/2 | 0 | -1/2 |
| 2 | 3 | 2 | | | 1 | -2 | 1 |
| | 3 | 2 | | 1 | -2 | 1 | |
| | 3 | 2 | 1 | -2 | 1 | | |

## Herleitung Newton

$$Q(x) \overset{!}{=} \min_{x \in \mathcal{S} \subset \mathbb{R}^n}$$

**Quadratisches Ersatzproblem:**

$$\tilde{Q}(x) = Q(x_k) + g_k^T \cdot (x - x_k) + \tfrac{1}{2}(x - x_k)^T \cdot H_k \cdot (x - x_k)$$

$$\frac{\delta \tilde{Q}}{\delta x} \overset{!}{=} 0_n \iff g_k + H_k \cdot (x - x_k)|_{x = x_{opt}} = 0 \implies x_{k+1} := x_{opt} = x_k - H_k^{-1} \cdot g_k$$

**Allgemeine Struktur des Algorithmus:**

$x_{k+1} = x_k - \varrho_k R_k g_k$

| | |
|---|---|
| $R_k = I_p$ | steilster Abstieg |
| $R_k = H_k^{-1}$ | Newton-Verfahren |

**Effiziente Variante:**

$$x_{k+1} = x_k - \Delta x_k, \qquad H_k \cdot \Delta x_k = -g_k .$$

## Fixpunktiteration

Kontraktionsbedingung:

$$K = \sup_{x \in U} \left\| \frac{\partial \Phi(x)}{\partial x^T} \right\| < 1$$

Konvergenzordnung:

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

p=1 linear
p=2 quadratisch
p=3 kubisch

Superlinear: c= {$c_k$} mit $c_k$ -> 0 für k -> inf

## Pseudocode tan x

```
double ownTangent(double x)
        x1 = (x mod pi) - pi/2
        if (abs(x1) <= ε) throw error
        else return tan(x)
```

Horner Schema programmieren!!!

## Pseudocode Lösen eines Dgl-Systems

[double[p,n], time[1,N]] ode(@f, double[p] initval, double start, double end, optional tol, optional verfahren)

## Runge-Kutta

$$y_1 = y_0 + \frac{h}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = f(x_0, y_0)$$

$$k_2 = f(x_0 + \tfrac{h}{2}, y_0 + \tfrac{h}{2} k_1)$$

$$k_3 = f(x_0 + \tfrac{h}{2}, y_0 + \tfrac{h}{2} k_2)$$

$$k_4 = f(x_0 + h, y_0 + h \cdot k_3)$$

## Numerische Integration

$$I_{Riemann} = h \cdot \sum_{i=0}^{n-1} f(x_i) \quad \text{mit} \quad h = \frac{b-a}{n}$$

$$I_{Trapez} = \frac{h}{2} \cdot \sum_{i=0}^{n-1} f(x_i) + f(x_{i+1})$$

$$I_{Simpson} = \frac{h}{6} \cdot \sum_{i=0, i gerade}^{n-2} f(x_i) + 4f(x_{i+1}) + f(x_{i+2})$$

## Newton-Raphson

(1) Löse das Lineare Gleichungssystem $J_k \Delta x_k = -f(x_k)$

(2) Setze $x_{k+1} = x_k + \Delta x_k$

J: Jacobimatrix

## Teststrategien für Algorithmen

- Numerische Problemfälle testen
- Schrittweite h->0 und Vergleich mit analytischer Lsg
- Fehlerhafte Eingaben
- Montecarlo Methoden
- Laufzeitmessung bei Echtzeitanwendungen

## Tikonov-Regularisierung

$$||Ax - b||_2^2 + \mu ||x||_2^2 \rightarrow Min$$

$LS - Lösung(ineffizient): x = (A^T A + \mu I_n)^{-1} A^T b$

Regularisierung stellt sicher, dass die Matrix positiv definit ist und vergrößert Spur, Determinante und Eigenwerte (µ > 0). Geometrisch bedeutet das, dass der Paraboloid stärker gewölbt ist, dass das Minimum also ausgeprägter erscheint, vgl. f(x) = $x^2$ und f(x) = $5x^2$.