3 Körper Problem

Von Lester Buhrmann und Dennis Budzinski

Inhaltsverzeichnis

Motivation

„So oft ich in den letzten 40 Jahren auch versucht habe, die Theorie der Mondbewegung aus den Grundlagen der Schwerkraft herzuleiten, haben sich immer so viele Schwierigkeiten aufgetürmt, dass ich gezwungen bin, meine letzten Forschungen abzubrechen.“

-Leonhard Euler

Motivation

- Problem: 3 Körper im Raum ergeben
  3 nichtlineare DGL 2. Ordnung
- Gekoppelte Gleichungen
- Nicht analytisch lösbar
- Stattdessen: Numerische Lösungsverfahren
- Problem: Fehleranfälligkeit

Motivation

- Problem: 3 Körper im Raum ergeben
  3 nichtlineare DGL 2. Ordnung
- Gekoppelte Gleichungen
- Nicht analytisch lösbar
- Stattdessen: Numerische Lösungsverfahren
- Problem: Fehleranfälligkeit

Lösungsverfahren

- Euler Verfahren
- Tangentenzerlegung:

$$f(x) = f'(x_o) \cdot (x - x_o) + f(x_o) + R_f(x, x_o)$$

- Restterm nichtlinear gegen 0
- $dt=t-t_o$ muss klein sein für max. Genauigkeit
- Je größer f"(t) (Kraft), desto ungenauer die Approximation

Lösungsideen

- Zeitintervall dt muss von Kraft abhängig sein
- Auch von Geschwindigkeit
- dt~F/v

Lösungsideen

- Einführung einer Vektor 3D Klasse
- Anlegen von arrays für Vektorgrößen
- Auch für ein array für dt (dt veränderlich!)
- Eulerverfahren in for-Schleife anwenden
- Kräfte, Geschwindigkeiten und Orte bei jeder Iteration aktualisieren
- Einführung von speziellen Abstandsfaktoren $D_{körper1,2}$ und $D_{körper2,3}$ …
- Dann ist dt=k(dt[0])*$D_{körpera,b}$, wenn $D_{körpera,b}$ zu klein

Konkrete Umsetzung

```
1  #ifndef VEKTOR_H_INCLUDED
2  #define VEKTOR_H_INCLUDED
3  class Vektor
4  {
5  public:
6    double x;
7    double y;
8    double z;
9
10   Vektor(double x, double y, double z);
11   Vektor operator+(const Vektor &b);
12
13 double operator*(const Vektor &a);
14
15 Vektor operator*(const double &a);
16
17 };
18 double norm(Vektor a);
19
20
21 #endif // VEKTOR_H_INCLUDED
```

Rechenoperationen für 3D Vektoren:

+:Addition
*:Skalarprodukt

Norm(): Betragsfunktion

- Skalarmultiplikation von rechts

```
125  vector<Vektor> xe, xm, xmerk, ve, vm, vmerk, ae, am, amerk;
126
127
128
129  xe.push_back(xe0);
130  xm.push_back(xm0);
131  xmerk.push_back(xmerk0);
132  ve.push_back(ve0);
133  vm.push_back(vm0);
134  vmerk.push_back(vmerk0);
135
136  vector<double> dt,t,k;
137
138  t.push_back(0);
139  dt.push_back(dt0);
140
141  vector<int> tn;
142
143  int N;
144  cout<<"Wähle die Anzahl der Berechnungsschritte(Empfehlung: 2
145  cin>>N;
146
147   double G = 6.67384e-11;
148   double m = -1;
```

- Vorbereitung von Arrays für die einzelnen Größen
- Eintragung der Anfangswerte
- Wahl der Anzahl Berechnungsdurchläufe(N)

Hier findet die Berechnung statt:

```
150    for(int i = 0;i <= N; ++i){
151   Vektor Rme = xm[i] + (xe[i]*m);
152   Vektor Remerk = xe[i] + (xmerk[i]*m);
153   Vektor Rmmerk = xm[i] + (xmerk[i]*m);
154   Vektor Fem = Rme*((mm*me)/((Rme*Rme)*norm(Rme)))*G;
155   Vektor Fmerke = Remerk*((mmerk*me)/((Remerk*Remerk)*norm(Remerk)))*G;
156   Vektor Fmerkm = Rmmerk*((mmerk*mm)/((Rmmerk*Rmmerk)*norm(Rmmerk)))*G;
157   ae.push_back(((Fmerke*m)+Fem)*(1/me)); //Beschleunigungsvektor der Erd
158   am.push_back((Fmerkm+Fem)*(m/mm));     //Beschhleunigungsvektor des Ma
159   amerk.push_back((Fmerke+Fmerkm)*(1/mmerk));
160   ve.push_back(ve[i]+(ae[i]*dt[i]));
161   vm.push_back(vm[i]+(am[i]*dt[i]));
162   vmerk.push_back(vmerk[i]+(amerk[i]*dt[i]));
163   xe.push_back(xe[i]+ve[i]*dt[i]);
164   xm.push_back(xm[i]+vm[i]*dt[i]);
165   xmerk.push_back(xmerk[i]+vmerk[i]*dt[i]);
166
167   t.push_back(t[i]+dt[i]);
168
169   k.push_back(round(t[i]*1/dt[0])*dt[0]);
```

Abstandsabhängiges Zeitintervall

Am Ende der for-Schleife wird die Größe von dt falls nötig verkleinert oder vergrößert:

```
174   double Dme = ((norm(Rme)*norm(Rme))/norm(vm[i]))*1e-7*dt[0];
175   double Demerk = ((norm(Remerk)*norm(Remerk))/norm(vmerk[i]))*1e-7*dt[0];
176   double Dmmerk = ((norm(Rmmerk)*norm(Rmmerk))/norm(vmerk[i]))*1e-7*dt[0];
177
178   if((dt[0] >= Dme)||(dt[0] >= Demerk)||(dt[0] >= Dmmerk))    //Kontrolle des
179   {
180    if((Dme < Dmmerk)&&(Dme < Demerk)){
181    dt.push_back(Dme);}                              //Funktion für veränderliches
182   if((Dmmerk < Dme)&&(Dmmerk < Demerk)){
183    dt.push_back(Dmmerk);}
184    if((Demerk < Dme)&&(Demerk < Dmmerk)){
185    dt.push_back(Demerk);}
186   }
187      else{
188        dt.push_back(dt[0]);                       //ansonsten zurück zum groben I
189                              }
190
```

Linearisierung der Zeitachse für Plots

```
193     for(int i = 1; i <= N; i++){  //
194     if(k[i] > k[i-1]){            //
195     tn.push_back(i);}
```

```
FILE* fp;
fp = fopen(xKoerper1.c_str(),"w");
for(int j = 0;j <= Schrittzahl; j++)
{


fprintf(fp,"%e %e %e\n", xe[tn[j]].x, xe[tn[j]].y, xe[tn[j]].z);


}
fclose(fp);
```

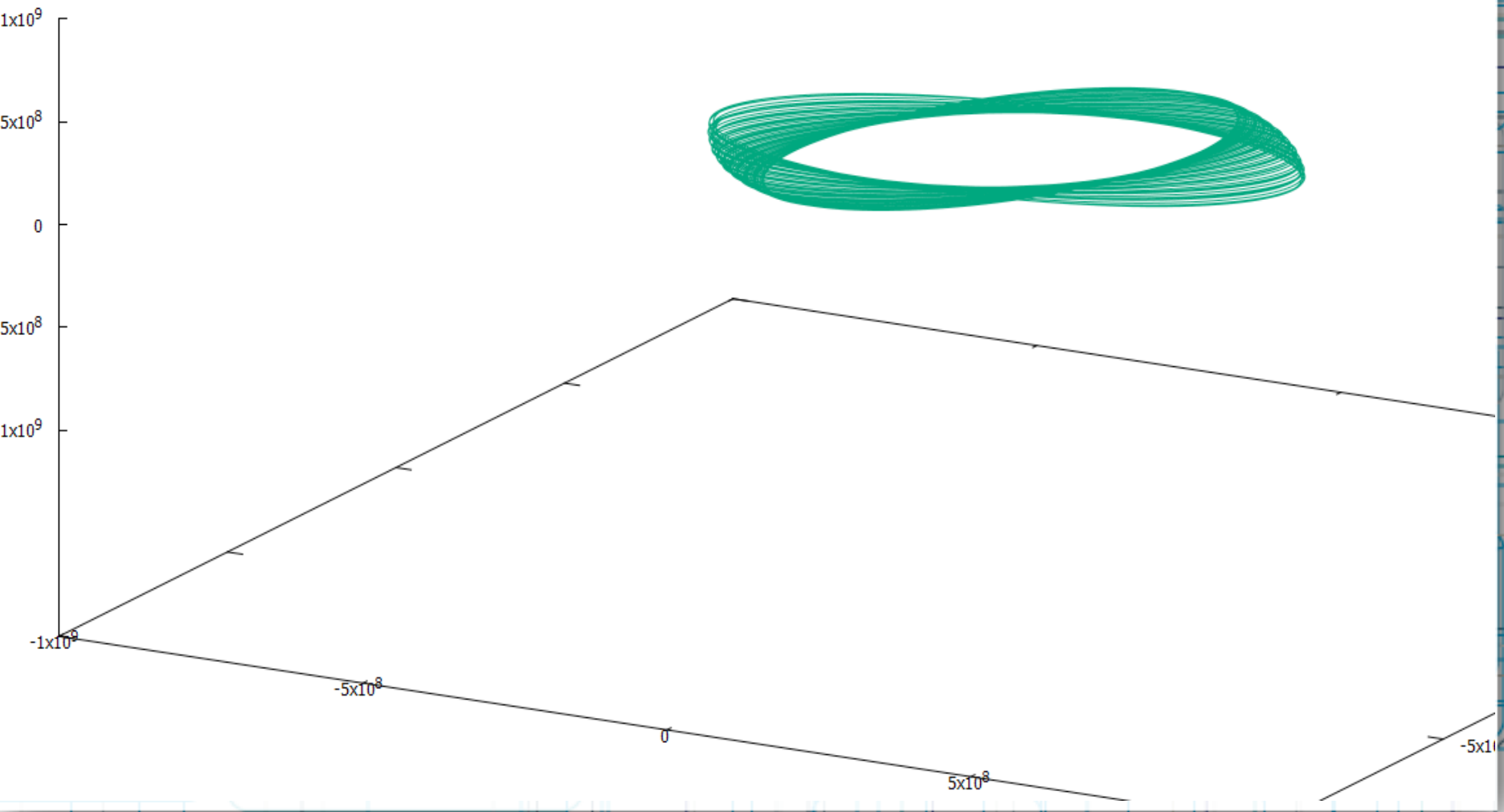
- Abstände der errechneten Werte zeitlich nicht gleich
- Für Erstellung der Text Datei für Plots werden nur die Werte verwendet, die um dt[0] entfernt sind

  In tn wird die Nummer aller zeitlich gleich entfernten Werte eingeschrieben

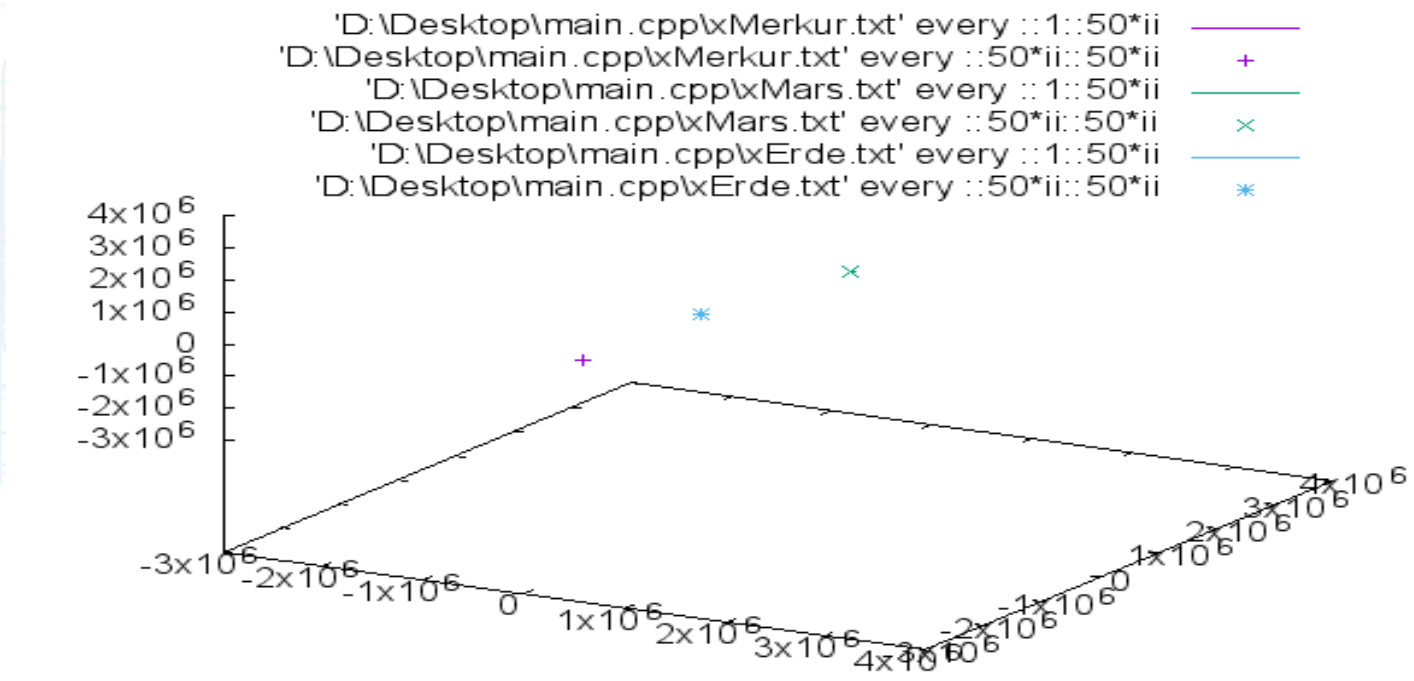
  Es gilt also:t[tn[j]]-t[tn[j-1]] ist immer dt[0]
- Deshalb wird hier tn[j] und nicht j eingesetzt

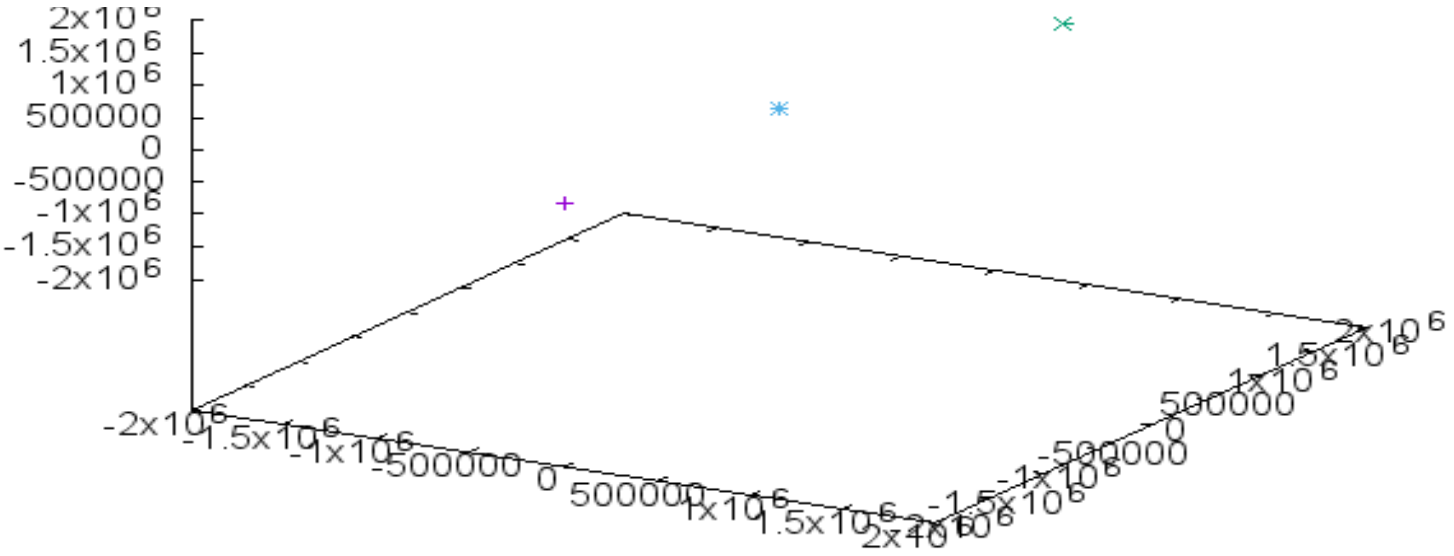
Resultate

Hier ist die Mondumlaufbahn
um die Erde innerhalb von
ungefähr 4 Jahren erkennbar:



Hier ist die Bewegung der Erde, des Mars und des Merkurs bei erfundenen Anfangsbedingungen zu sehen:





Ausblick

Erweiterung auf N Körper

- Da die Intervalle bereits im Programm ausgerechnet werden, lohnt es sich, die Plots direkt zu produzieren
- Ein Programm bauen, dass habitable Orbitale für Planeten in 2-Sonnen Systemen findet
- Weitere Auslagerung von Funktionen(Übersichtlichkeit)