Gruppe 1
Freitag 13:15-14:45, B-Woche
(Dr. B. Schulze)

Claudia Heinich, 8966108
Sven Pohl, 8968075
Arne Brutschy, 8964813

Numerik

Serie 11

1. a) Hier die Ergebnisse unseres Algorithmus:

i	$f(i)$	Splineinterpolation	Differenz
0	0.04244032	0.04245889	-0.00001858
1	0.05245902	0.05196004	0.00049898
2	0.06639004	0.06576430	0.00062574
3	0.08648649	0.08542671	0.00105978
4	0.11678832	0.11500388	0.00178444
5	0.16494845	0.16164949	0.00329896
6	0.24615385	0.24012229	0.00603155
7	0.39024390	0.37812658	0.01211732
8	0.64000000	0.62814060	0.01185940
9	0.94117647	0.95546485	-0.01428838
10	0.94117647	1.00000000	-0.05882353
11	0.64000000	0.64453514	-0.00453514
12	0.39024390	0.37185945	0.01838445
13	0.24615385	0.23725784	0.00889601
14	0.16494845	0.15987843	0.00507003
15	0.11678832	0.11420989	0.00257843
16	0.08648649	0.08500615	0.00148033
17	0.06639004	0.06547922	0.00091082
18	0.05245902	0.05202259	0.00043643

Wie man deutlich sieht ist die Splineinterpolation deutlich genauer und gleichmäßiger als die Newtonsche. Vor allem bei hohen Potenzen war die Newtonsche schlechter, was bein der Splineinterpolation durch viele kleine kubische Polynome vermieden ist.

b) Hier der zugehörige Quellcode. Es ist zu beachten, dass das Programm LaTeXCode ausgibt. Wir verwenden den Gaussalgorithmus aus Aufgabe 16 um die tridiagonale Matrix zu lösen.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>

#define MAX 20

typedef struct {
  long double t[MAX];
  long double f[MAX];
  long double a[MAX];
  long double b[MAX];
  long double c[MAX];
  long double d[MAX];
  long double h[MAX];
```

Claudia Heinich, 8966108
Sven Pohl, 8968075
Arne Brutschy, 8964813

```
} spline_t;

typedef struct {
  double A[MAX][MAX];
  double b[MAX];
  double x[MAX];
  int n;
} gleichung_t;


void zeilentausch(gleichung_t *gl, int a, int b) {
  double tmp;
  int i;

  if (a == b) return;

  for (i= 0; i < gl->n; i++) {
    tmp= gl->A[a][i];
    gl->A[a][i]= gl->A[b][i];
    gl->A[b][i]= tmp;
  }

  tmp= gl->b[a];
  gl->b[a]= gl->b[b];
  gl->b[b]= tmp;

  return;
}

void gauss(gleichung_t *gl) {
  int i, j, k;
  double l;

  for (i= 0; i < gl->n-1; i++) {
    // suche pivotelement
    j= i;
    while (gl->A[j][i] == 0 && j < gl->n) j++;

    if (gl->A[j][i] == 0) {
      printf("Matrix ist singulaer!\n");
      return;
    }
    zeilentausch(gl, i, j);

    // jede Zeile
    for (k= i + 1; k < gl->n; k++) {
      l= gl->A[k][i]/gl->A[i][i];

      // Zeile multiplizieren
```

Claudia Heinich, 8966108
Sven Pohl, 8968075
Arne Brutschy, 8964813

```
      for (j= i; j < gl->n; j++)
        gl->A[k][j]= gl->A[k][j] - l * gl->A[i][j];

      gl->b[k]= gl->b[k] - l * gl->b[i];
    }
  }

  // errechne ergebnis
  for (i= gl->n-1; i >= 0; i--) {
    // errechne Wert fuer diese Zeile
    gl->x[i]= gl->b[i];
    for (j= i+1; j < gl->n; j++) {
      gl->x[i]-= gl->A[i][j];
    }
    gl->x[i]= gl->x[i]/gl->A[i][i];

    for (j= i+1; j >= 0; j--)
      gl->A[j][i]= gl->A[j][i] * gl->x[i];
  }
  return;
}


void spline(spline_t *s) {
  int i;
  gleichung_t gl;

  // h_i und a_i berechnen
  for (i = 0; i < MAX - 1; i++) {
    s->h[i] = s->t[i+1] - s->t[i];
    s->a[i] = s->f[i];
  }
  s->a[MAX-1] = s->f[MAX-1];

  s->c[0] = 0;
  s->c[MAX-1] = 0;

  // initalisiere tridiagonale Matrix
  memset(&gl,  0, sizeof(gl));
  for (i = 0; i < MAX-2; i++) {
    gl.A[i][i] = 2*(s->h[i] + s->h[i+1]);
    gl.A[i+1][i] = s->h[i];
    gl.A[i][i+1] = s->h[i];
    gl.b[i] = 3/s->h[i+1]*(s->a[i+2] - s->a[i+1]) -
              3/s->h[i]*(s->a[i+1] - s->a[i]);
  }

  // berechne c_i Matrix mittels Gauss ohne Pivotsuche
  gl.n = MAX-2;
```

Claudia Heinich, 8966108
Sven Pohl, 8968075
Arne Brutschy, 8964813

```
  gauss(&gl);
  // kopiere Werte um
  for (i = 0; i < MAX-2; i++)
    s->c[i+1] = gl.x[i];

  // berechne b_i und d_i
  for (i = 0; i < MAX; i++) {
    s->b[i] = 1/s->h[i]*(s->a[i+1] - s->a[i]) -
              s->h[i]/3*(s->c[i+1] + 2*s->c[i]);
    s->d[i] = 1/(3*s->h[i])*(s->c[i+1] - s->c[i]);
  }
}

long double calc_spline(spline_t *s, long double t) {
  int i;

  for (i = 0; t > s->t[i]; i++);
  i--;

  return s->a[i] + s->b[i]*(t - s->t[i]) + 1/2*s->c[i] *
           pow((t - s->t[i]), 2) + 1/6*s->d[i]*pow((t - s->t[i]), 3);
}

long double calc_f(long double t) {
  return 1/(1 + pow(t, 2));
}

int main() {
  spline_t s;
  long double t, s2, f;
  int i;

  // berechne t_i und f_i
  for (i = 0; i < MAX; i++) {
    s.t[i] = -5 + (10.0*i)/MAX;
    s.f[i] = calc_f(s.t[i]);
  }

  spline(&s);

  for (i = 0; i < MAX-1; i++) {
    t = s.t[i] + (s.t[i+1] - s.t[i]) / 2.0;
    s2 = calc_spline(&s, t);
    f = calc_f(t);
    printf("          %2d & %.8Lf & %.8Lf & %.8Lf\\\\\n", i, f, s2, f - s2);
  }

  return EXIT_SUCCESS;
}
```

2. a) Hier die Ergebnisse unseres FORTRAN77 Programmes:

i	h	$f_1(t,h)$	Differenz f_1	$f_2(t,h)$	Differenz f_2
1	0.100000001	-6.27334572	-2.63007785	-10.2987277	1.39530417
2	0.0100000007	-8.55909966	-0.344323908	-8.91584754	0.0124239746
3	0.00100000005	-8.86791358	-0.0355099909	-8.90354767	0.000124103826
4	0.000100000012	-8.89986144	-0.00356212808	-8.90342481	1.24102263E-06
5	1.00000016E-05	-8.90306724	-0.000356324667	-8.90342358	1.23707302E-08
6	1.00000022E-06	-8.90338793	-3.56360681E-05	-8.90342357	-5.81511728E-10
7	1.00000015E-07	-8.90341999	-3.57614402E-06	-8.90342357	-1.22643051E-09
8	1.00000026E-08	-8.90342306	-5.1051742E-07	-8.90342359	2.23894929E-08
9	1.0000003E-09	-8.90342284	-7.28273081E-07	-8.90342462	1.04808322E-06
10	1.00000036E-10	-8.90340813	-1.54333617E-05	-8.9034259	2.33020024E-06
11	1.00000034E-11	-8.90327506	-0.00014850563	-8.9034527	2.91299935E-05
12	1.00000043E-12	-8.9013203	-0.00210326955	-8.90309665	-0.000326913473
13	1.00000039E-13	-8.87289896	-0.030524606	-8.89066252	-0.0127610445
14	1.00000049E-14	-8.79296204	-0.110461526	-8.97059764	0.0671740703
15	1.00000043E-15	-7.99360236	-0.909821204	-9.76995844	0.866534877
16	1.00000055E-16	8.88177935	-17.7852029	-8.88177935	-0.0216442218
17	1.00000056E-17	0.	-8.90342357	0.	-8.90342357
18	1.00000067E-18	0.	-8.90342357	0.	-8.90342357
19	1.00000061E-19	0.	-8.90342357	0.	-8.90342357
20	1.0000007E-20	0.	-8.90342357	0.	-8.90342357

Das Ergebnis der symbolischen Differentiation ist -8.90342357. Mit diesem Referenzergebnis werden die Ergebnisse der numerischen Approximationen verglichen. Für grosse i ist $h^{-i} = 0$ was die Ergebnisse von 17 bis 20 erklärt.
Ansonsten ist die Varianz des zweiten Approximationsverfahrens grösser, wobei für grössere i das erste Approximationsverfahren genauer ist.

b) Der zugehörige Quellcode. Es ist wiederum zu beachten, daß das Programm LaTeXCode ausgibt.

```
PROGRAM aufgabe22
DOUBLE PRECISION  F,DF,APP1,APP2,X,H,ABL,AP1,AP2
INTEGER i
X=0.3
ABL=DF(X)
WRITE(*,*) 'Ableitung:',ABL
DO i=1,20,+1
  H=10.0**(i*-1)
  AP1=APP1(X,H)
  AP2=APP2(X,H)
  WRITE(*,*) '           ',i,' & ',H,' & ',AP1,' & ',ABL-AP1
  WRITE(*,*) ' & ', AP2, ' & ', ABL-AP2, ' \\\\'

END DO
end
```

Claudia Heinich, 8966108
Sven Pohl, 8968075
Arne Brutschy, 8964813

```
FUNCTION F(T)
IMPLICIT REAL*8(A-H,O-Z)

U=DEXP(DSIN(T))
V=U-T**2
W=U*V/T
F=DLOG(T+U)+W
RETURN
END

FUNCTION DF(T)
IMPLICIT REAL*8(A-H,O-Z)
DT=1.0
DU=DEXP(DSIN(T))*DCOS(T)*DT
U=DEXP(DSIN(T))
DV=DU-(2.0*T)
V=U-T**2.0
DW=((U*DV+DU*V)*T-U*V*DT)/T**2.0
W=U*V/T
DF=(1.0/(T+U))*(DT+DU)+DW
F=DLOG(T+U)+W
RETURN
END FUNCTION

DOUBLE PRECISION FUNCTION APP1(T,H)
IMPLICIT DOUBLE PRECISION(A-H,O-Z)
  APP1=(F(T+H)-F(T))/H
RETURN
END FUNCTION

DOUBLE PRECISION FUNCTION APP2(T,H)
IMPLICIT DOUBLE PRECISION(A-H,O-Z)
APP2=(F(T+H)-F(T-H))/(2*H)
RETURN
END FUNCTION
```