A STUDY OF INDUCTIVE AND CAPACITIVE IGNITION AND KERNEL DEVELOPMENT

Diploma paper

by

Rikard Carlsson
and
Bengt Johnsson

Contents

1. Abstract

One way of improving the performance of an internal combustion engine is to improve the ignition. The aim of this study was to investigate how the flame speed in a burning gas is influenced by the electrical properties of the spark. This was done by recording the voltage, current and flame front of single sparks and comparing them individually and statistically. Two different ignition systems were used, one based on an inductive discharge and one on a capacitive discharge. The explosive gas was a mixture of propane and air, just above stoichiometry. The measurements were performed at atmospheric pressure and at 4 bar. Two different types of spark plugs were used: one with steel electrodes and one with copper electrodes.

2. Atmospheric pressure measurements

2.1 Introduction

Although it may seem advantageous to study the process under conditions normally found in an engine, carrying out the experiments at atmospheric pressure has some important advantages. Firstly, it is a fast way of making measurements. No filling of high-pressure cells is needed, no time-wasting opening or closing of valves etc. Therefore , it is possible to observe a large number of sparks at each parameter setting, which will reduce statistical errors. Secondly, the behaviour of flames under atmospheric conditions, i.e. flame front speed etc, is well known for all air/fuel ratios.

2.2 Experimental Details

The optical arrangement can be seen in fig. 2.1 and the electrical arrangement in figs 2.2 - 2.4. From the gas tubes the gases are premixed using two flow meters. The air/fuel mixture then passes through a glass tube, as shown in fig. 2.5. An almost perfectly homogeneous velocity distribution is provided by the sintered glass disc. The spark plug is positioned about an inch above the tube opening. (For optical reasons it must be *above*, but to obtain smaller cycle-to-cycle variations we would prefer it to be *in* the tube). Once the gas is ignited, it will continue to burn until extinguished by a pressurized air blast, controlled by a magnetic valve.

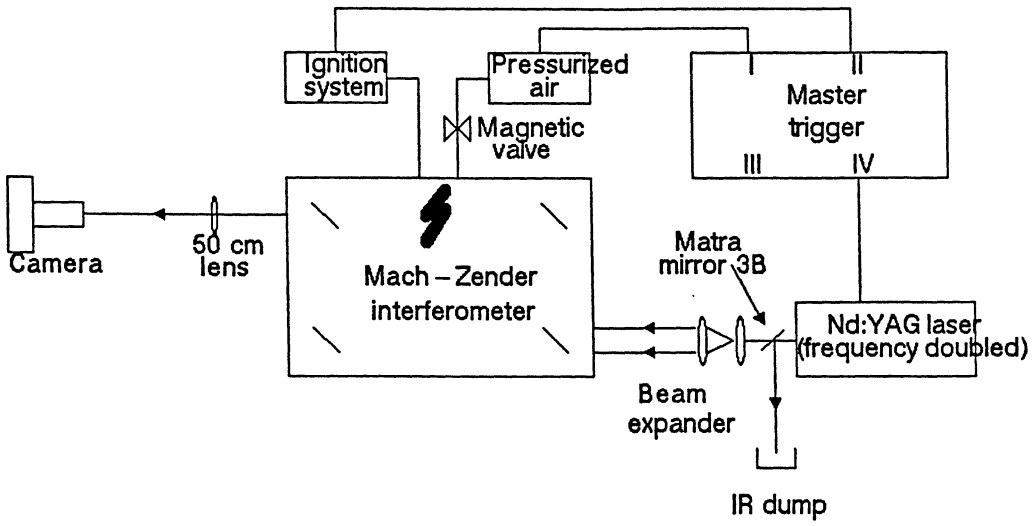


Fig. 2.1 The optical arrangement

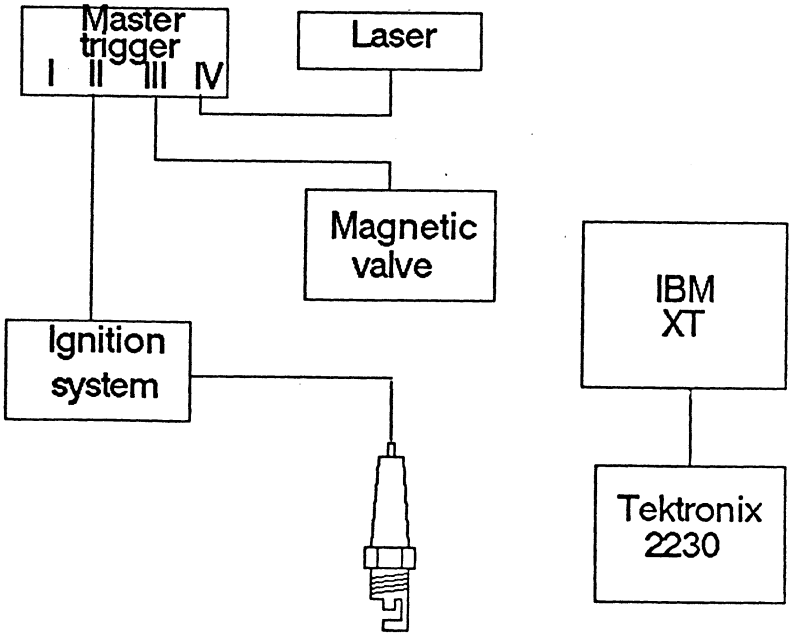


Fig. 2.2 The electrical arrangement

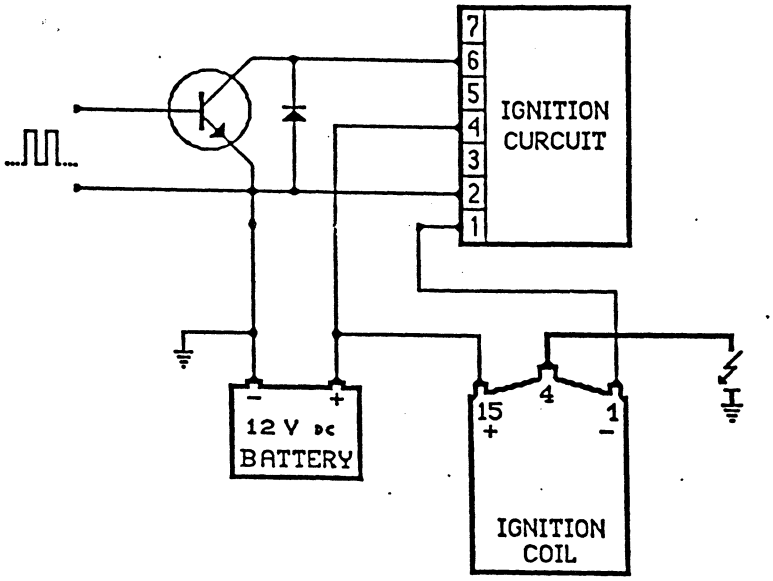


Fig. 2.3 The inductive system

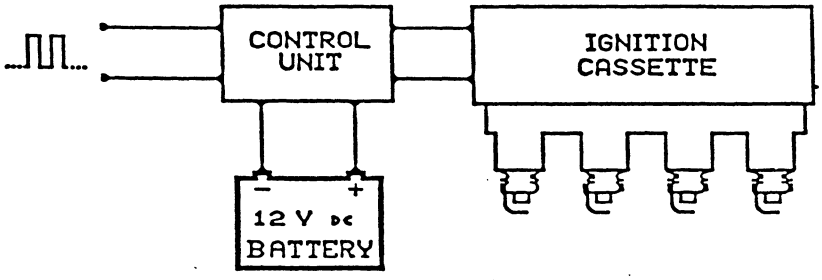


Fig. 2.4 The capacitive system

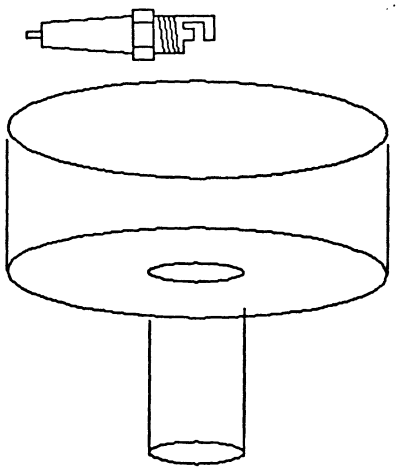
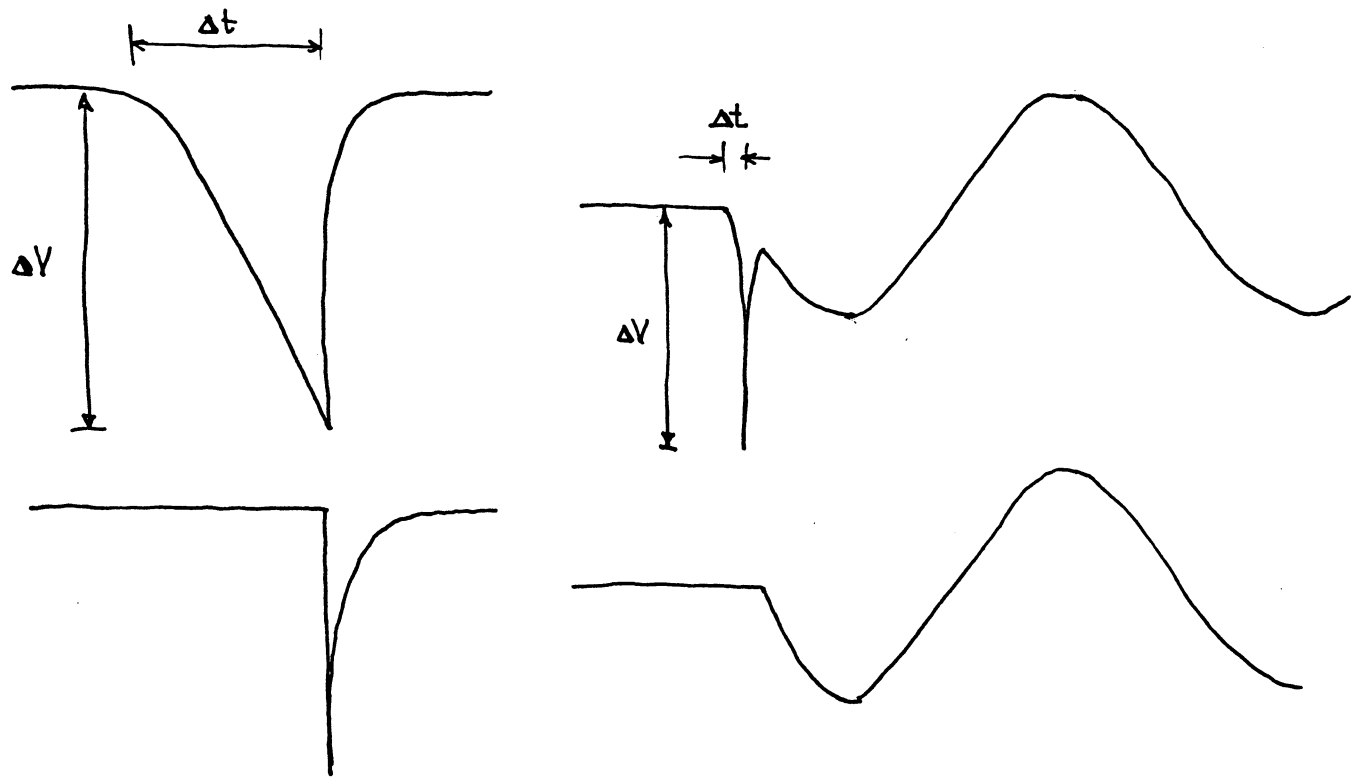


Fig. 2.5 The glass tube

The voltage over and current through the spark plug were measured with a Tektronix 2230 digital storage oscilloscope and a Tektronix P 6015 high-voltage probe. Stored data were transferred to an IBM XT computer for processing, as described in Appendices A and B.

We chose to record the current and voltage at the 10 μs/div setting of the oscilloscope as this was needed to resolve the fast voltage ramp, but still gave the opportunity to observe the whole of the interesting time region. According to the literature, the most important part of the energy lies in this region (see Discussion). This means that the curves recorded on the oscilloscope appeard as those shown in fig. 2.6 with ΔV and Δt (i.e. the breakdown voltage and the time lag to breakdown after the voltage is applied) shown. Typical values of ΔV and Δt at atmospheric pressure are 8.5 kV and 30 μs for the inductive system, and 10 kV and 2,5 μs for the capacitive system. The spark energy is shown in diagram 2.7 and 2.8 and the power level in diagram 2.9.



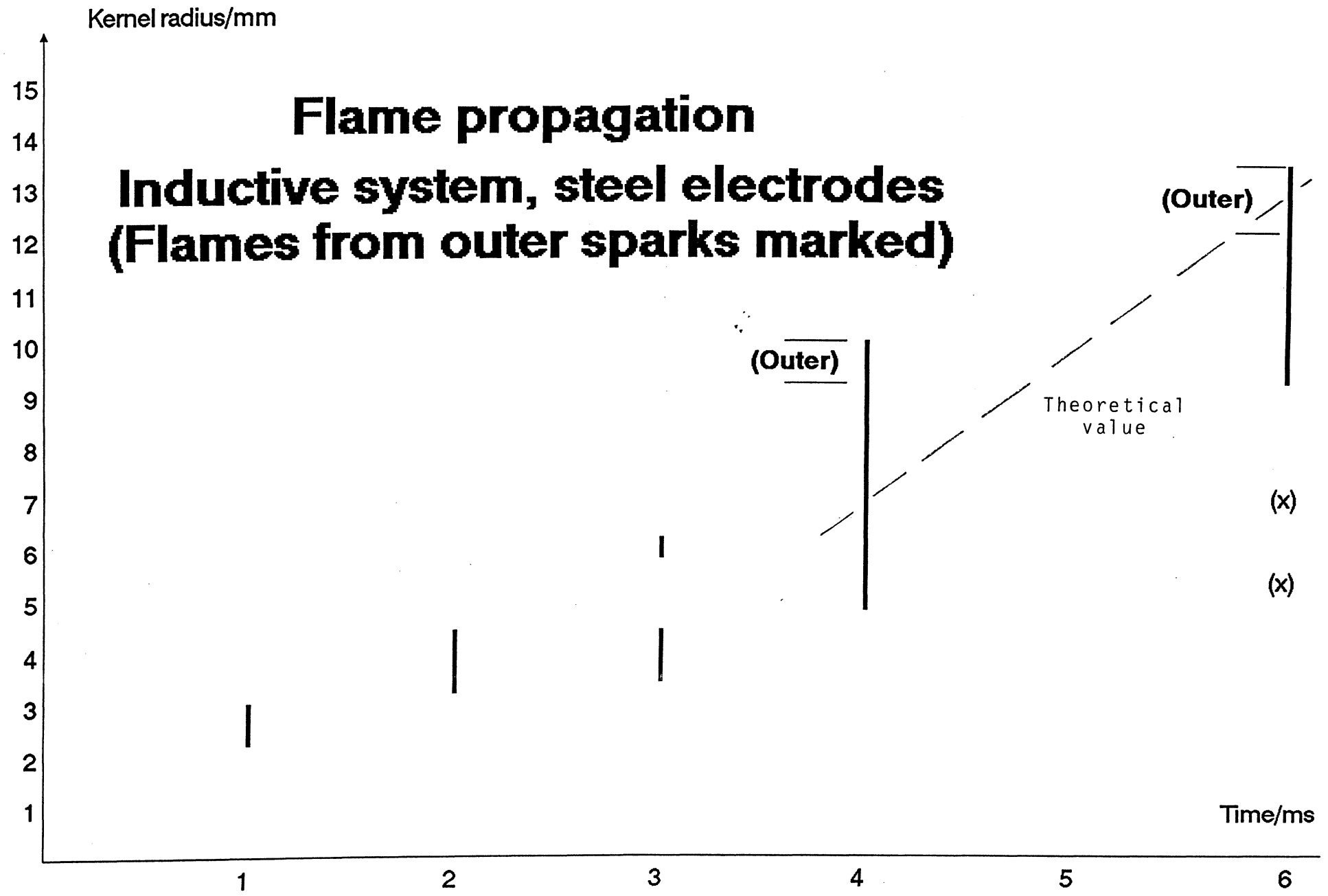
Inductive system

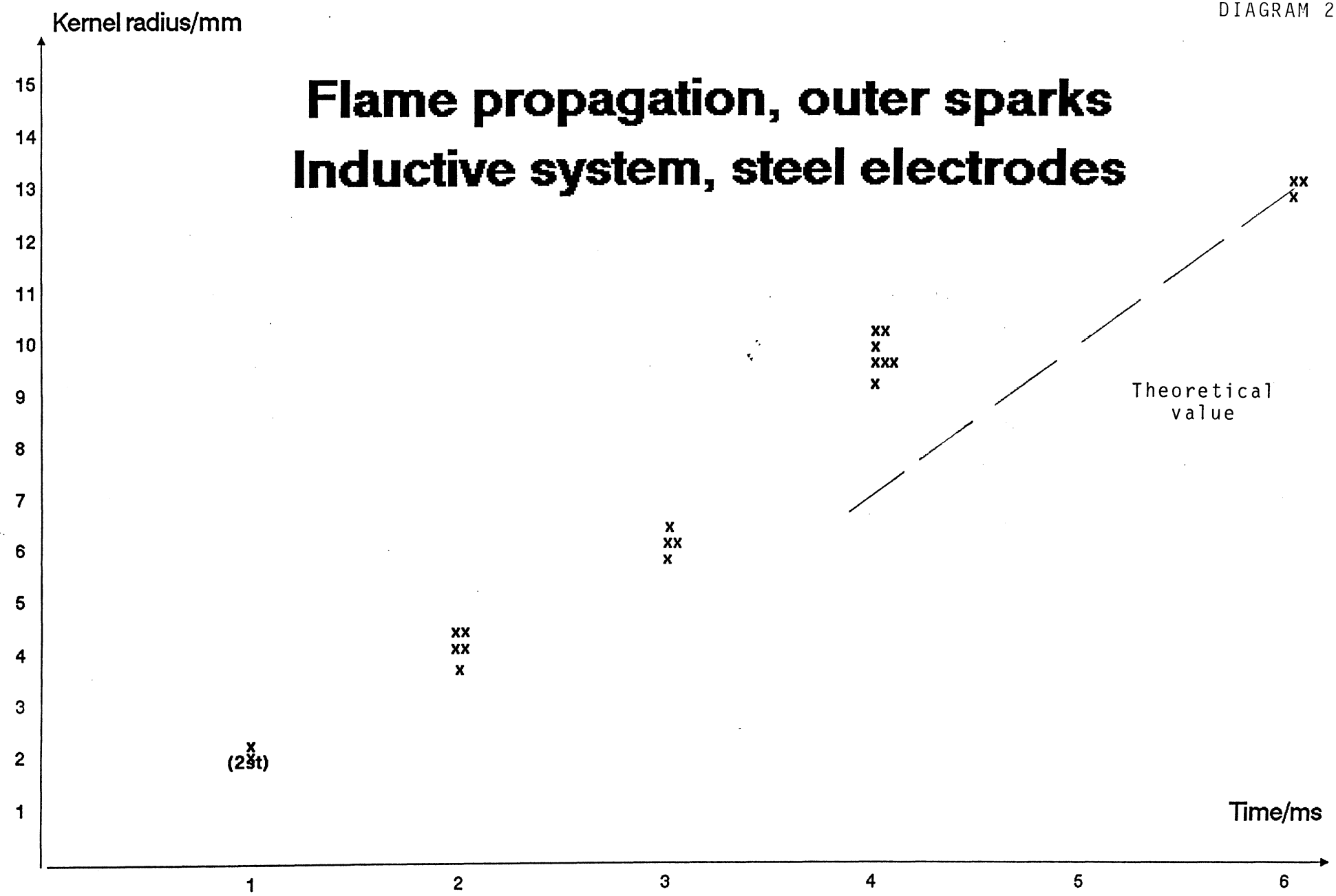
Capacitive system

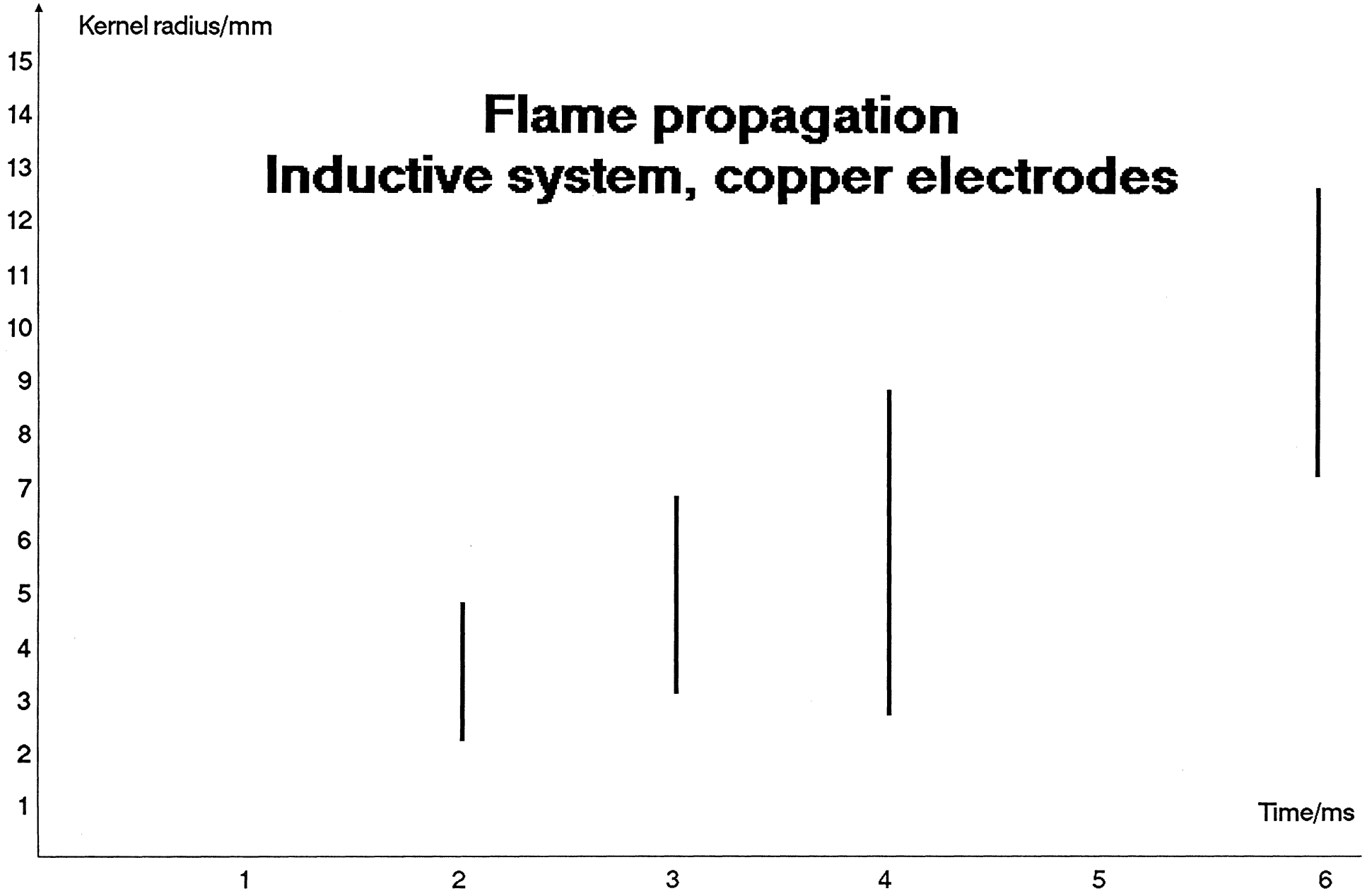
Fig. 2.6 The oscilloscope curves

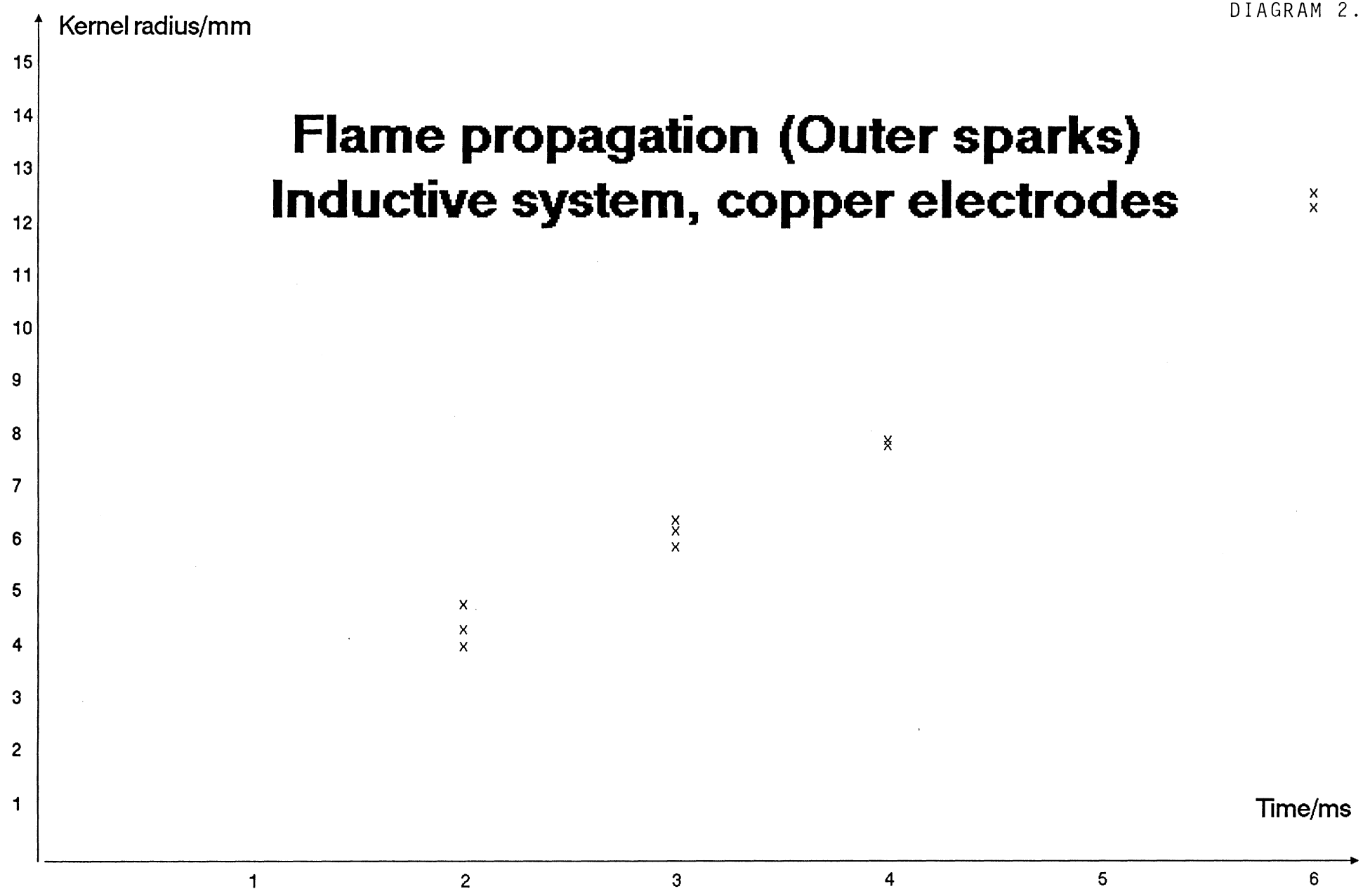
For optical measurements, a pulsed Nd:YAG laser, working at a wavelength of 1.064 μm, was controlled by the master trigger. A frequency doubling crystal gave green light at a wavelength of 532 nm, and remaining infra-red light was dumped using a Matra mirror 3B. Since the refractive index of a burning gas is very different from a gas at room temperature, the resulting interferogram from the Mach-Zender interferometer clearly indicates the position of the flame front at different time delays. A single 50 cm lens and a Nikon SLR 35 mm camera with a 300 mm lens were used to record the interferogram. The delays of the laser trigger signal were measured with a fast responding photodiode and the oscilloscope, comparing the output from the diode with a signal directly from the trigger. To evaluate the interferograms, an ordinary Abbe comparator was used. The exact value of the magnification is very sensitive to lens and camera positioning. There is also a certain magnification due to the shrinkage of the film. The total effect of these two phenomena can hardly be calculated, but is easily measured by making an exposure of a vernier cuplier set at 10.0 mm.

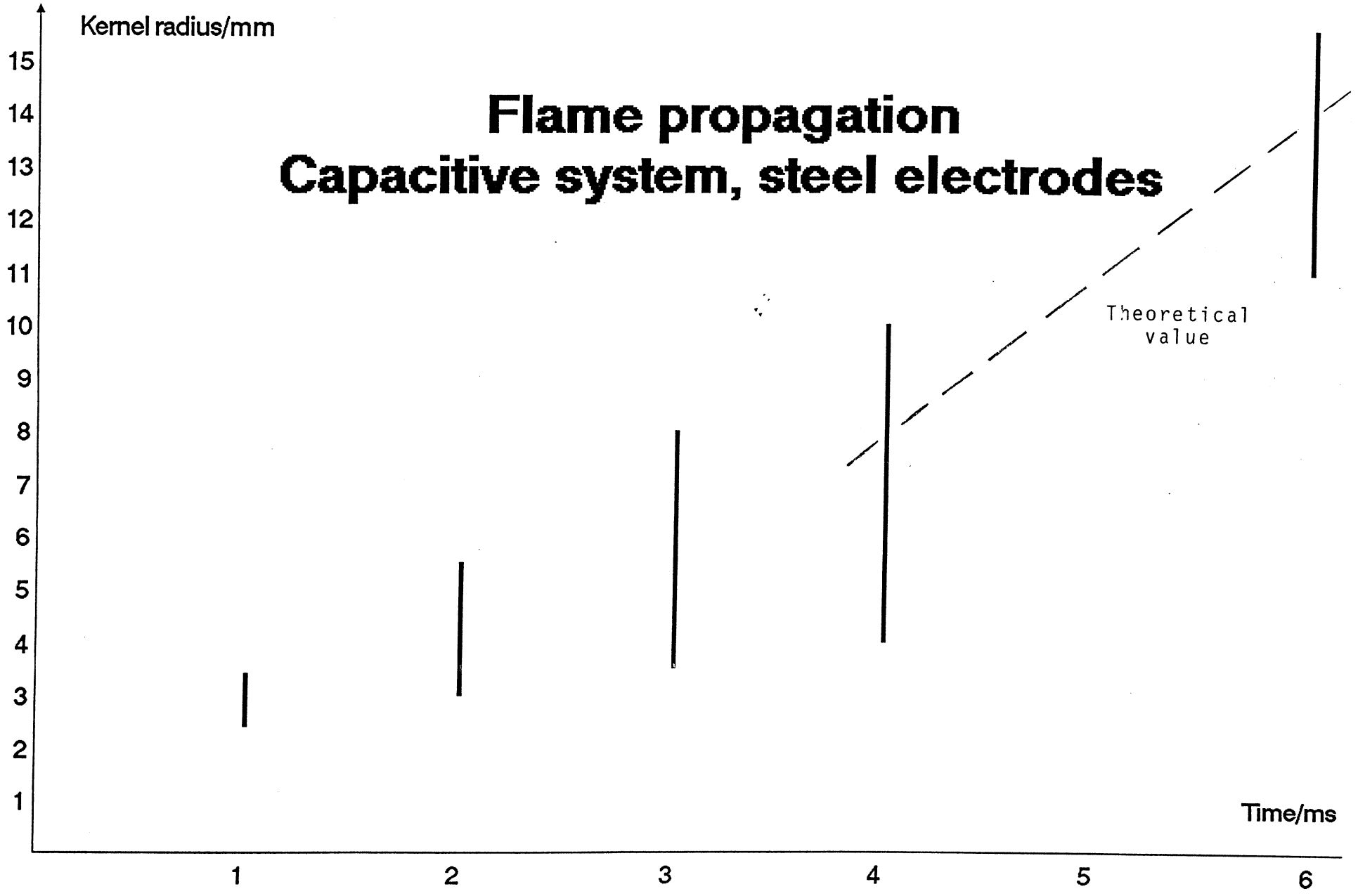
2.3 Diagrams


Kernel radius/mm
Flame propagation
Inductive system, steel electrodes
(Flames from outer sparks marked)
(Outer)
(Outer)
Theoretical value
(x)
(x)
Time/ms
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
1
2
3
4
5
6


Kernel radius/mm
Flame propagation, outer sparks
Inductive system, steel electrodes
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
(2st)
Theoretical value
Time/ms
1
2
3
4
5
6


Kernel radius/mm
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
Flame propagation
Inductive system, copper electrodes
Time/ms
1
2
3
4
5
6


Kernel radius/mm
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
Flame propagation (Outer sparks)
Inductive system, copper electrodes
Time/ms
1
2
3
4
5
6


Kernel radius/mm
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
Flame propagation
Capacitive system, steel electrodes
Theoretical value
Time/ms
1
2
3
4
5
6

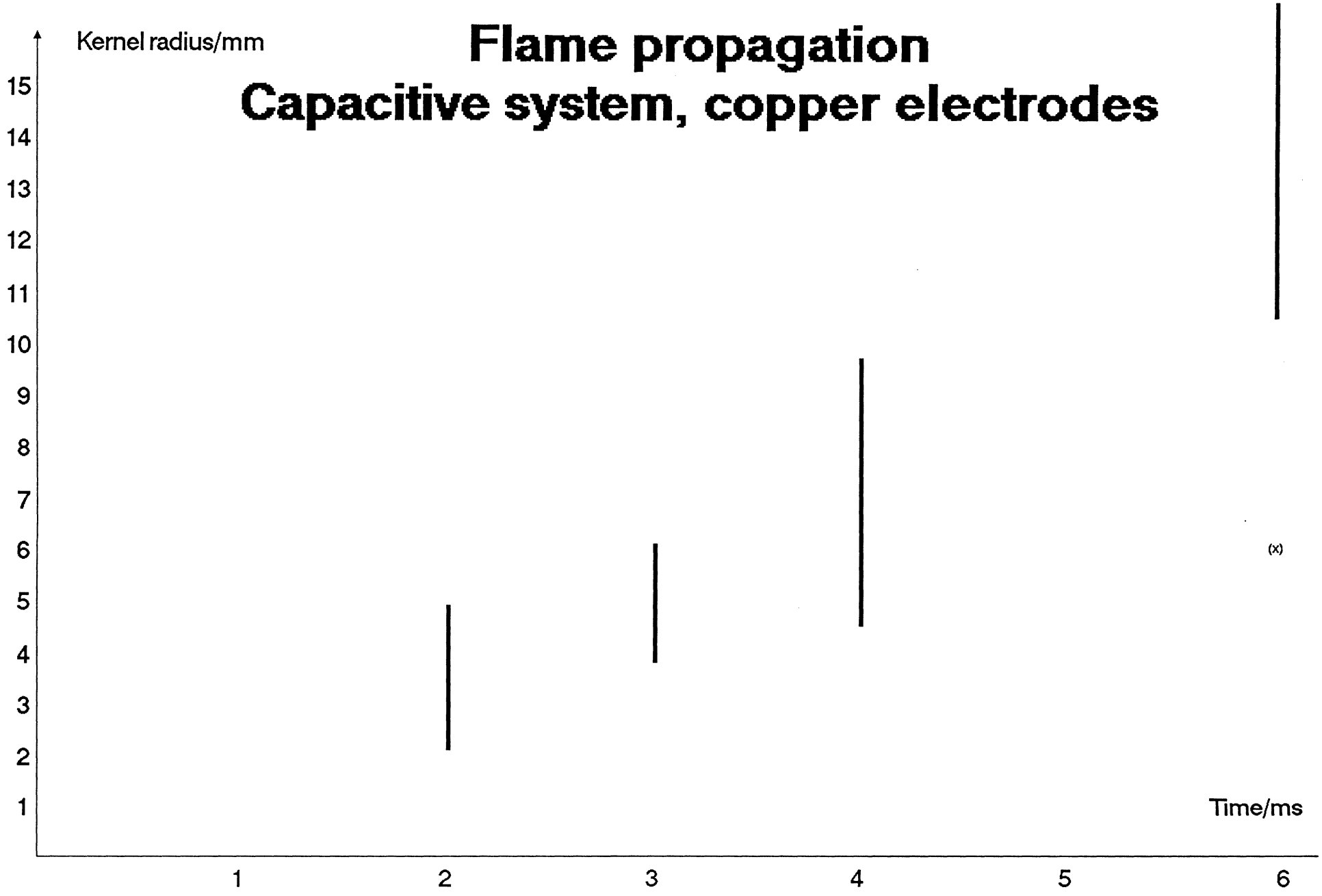
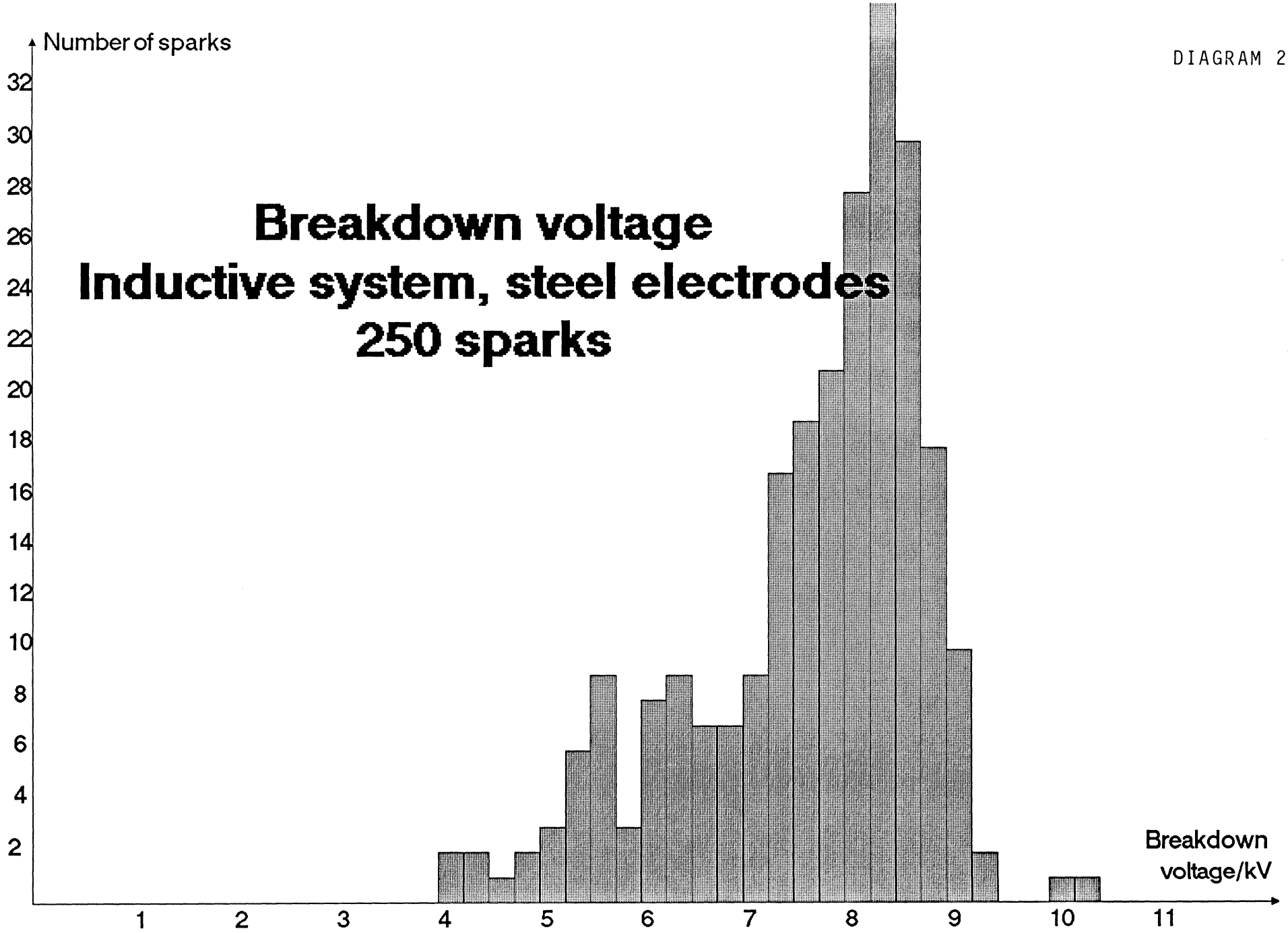

Kernel radius/mm
Flame propagation
Capacitive system, copper electrodes
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
(x)
Time/ms
1
2
3
4
5
6

DIAGRAM 2.3.2 A



Breakdown voltage
Inductive system, steel electrodes
Outer sparks

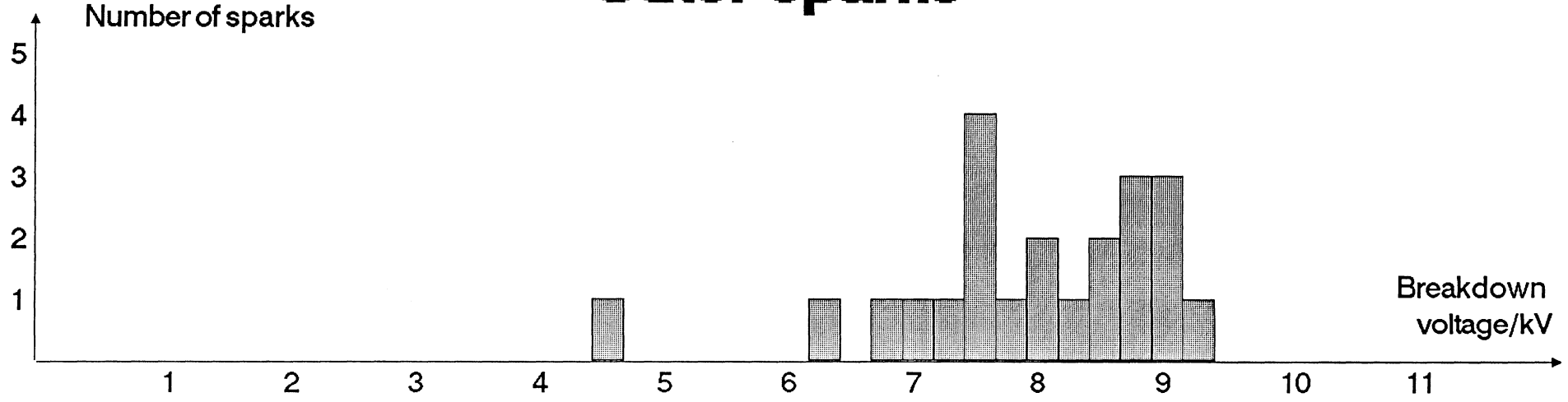
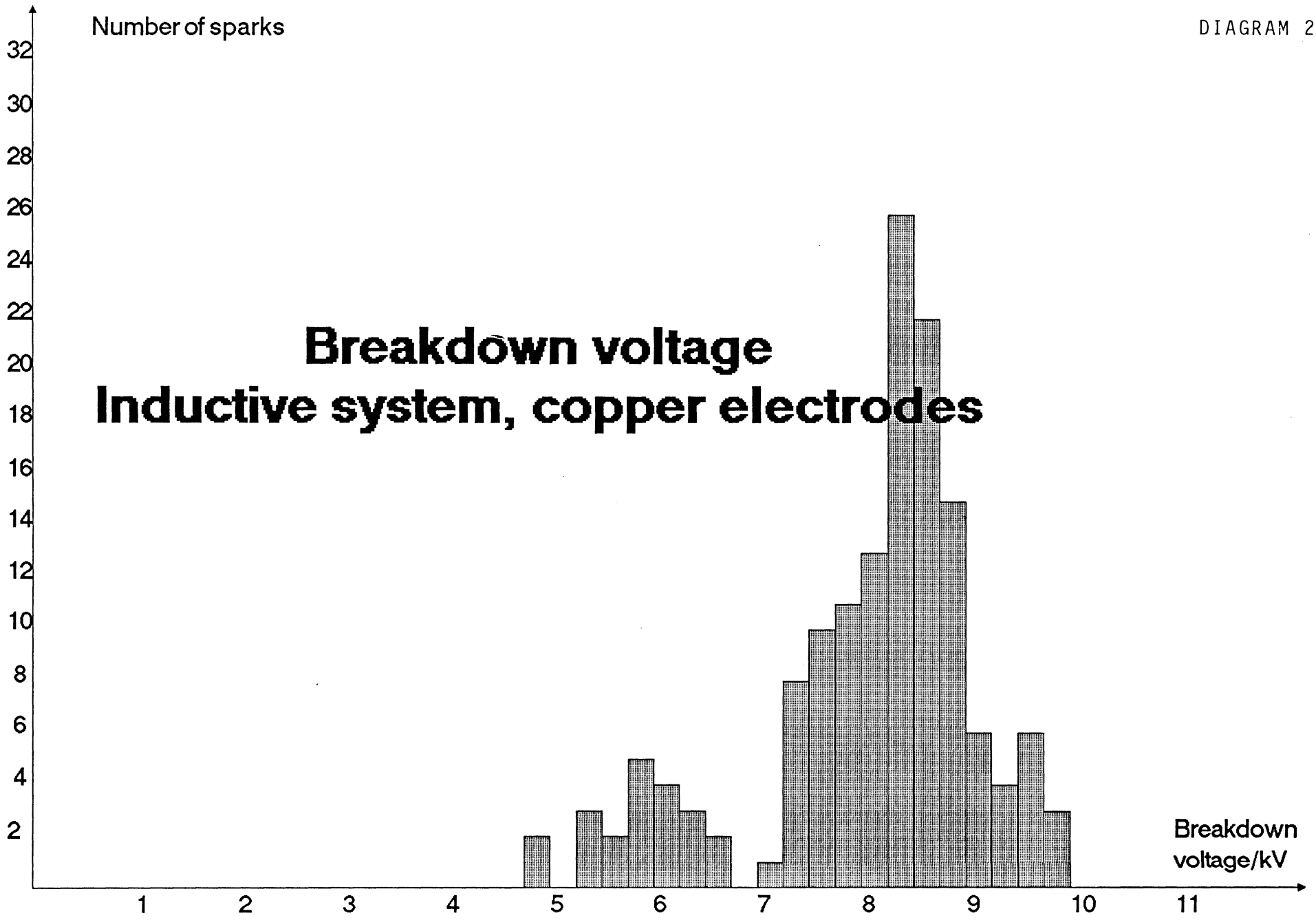
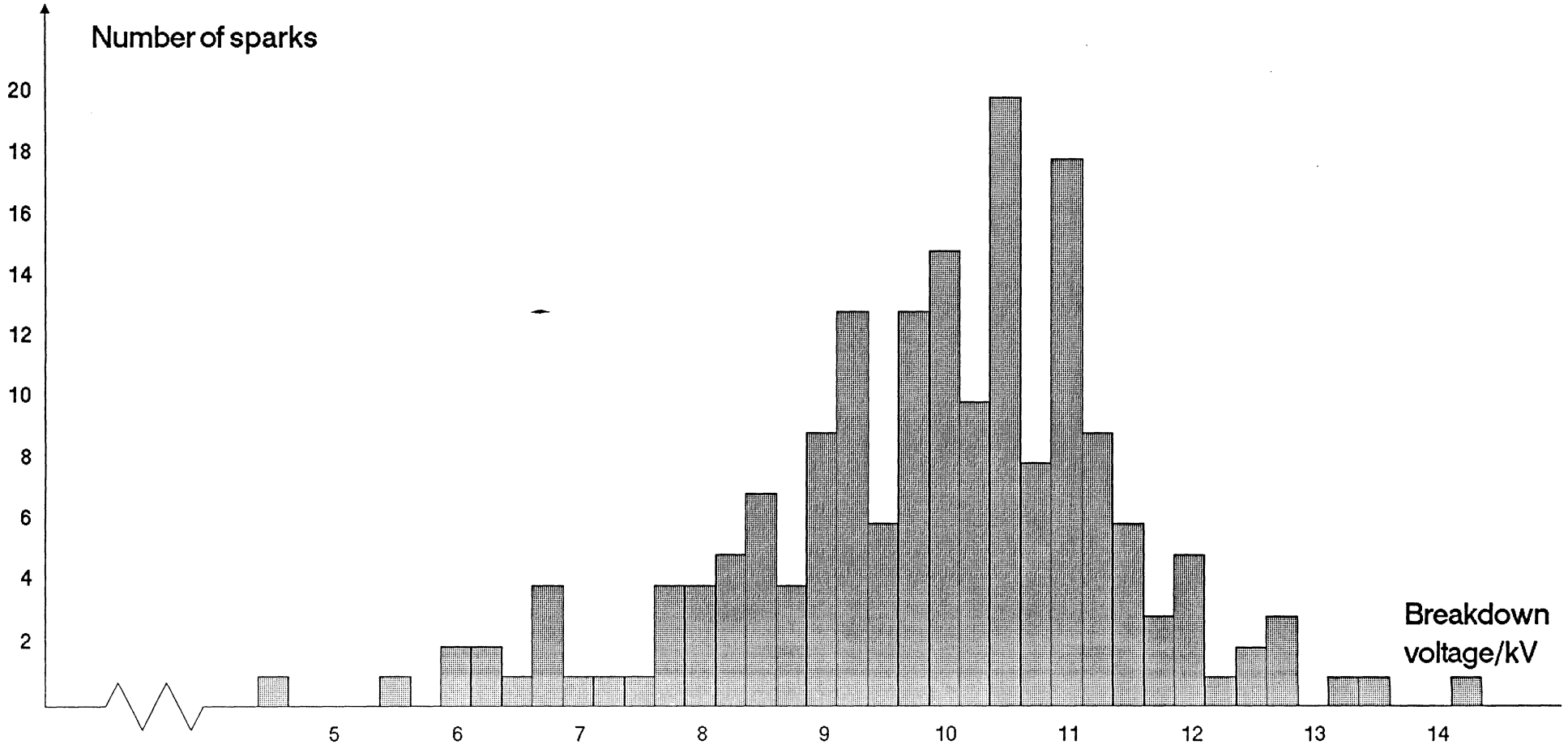
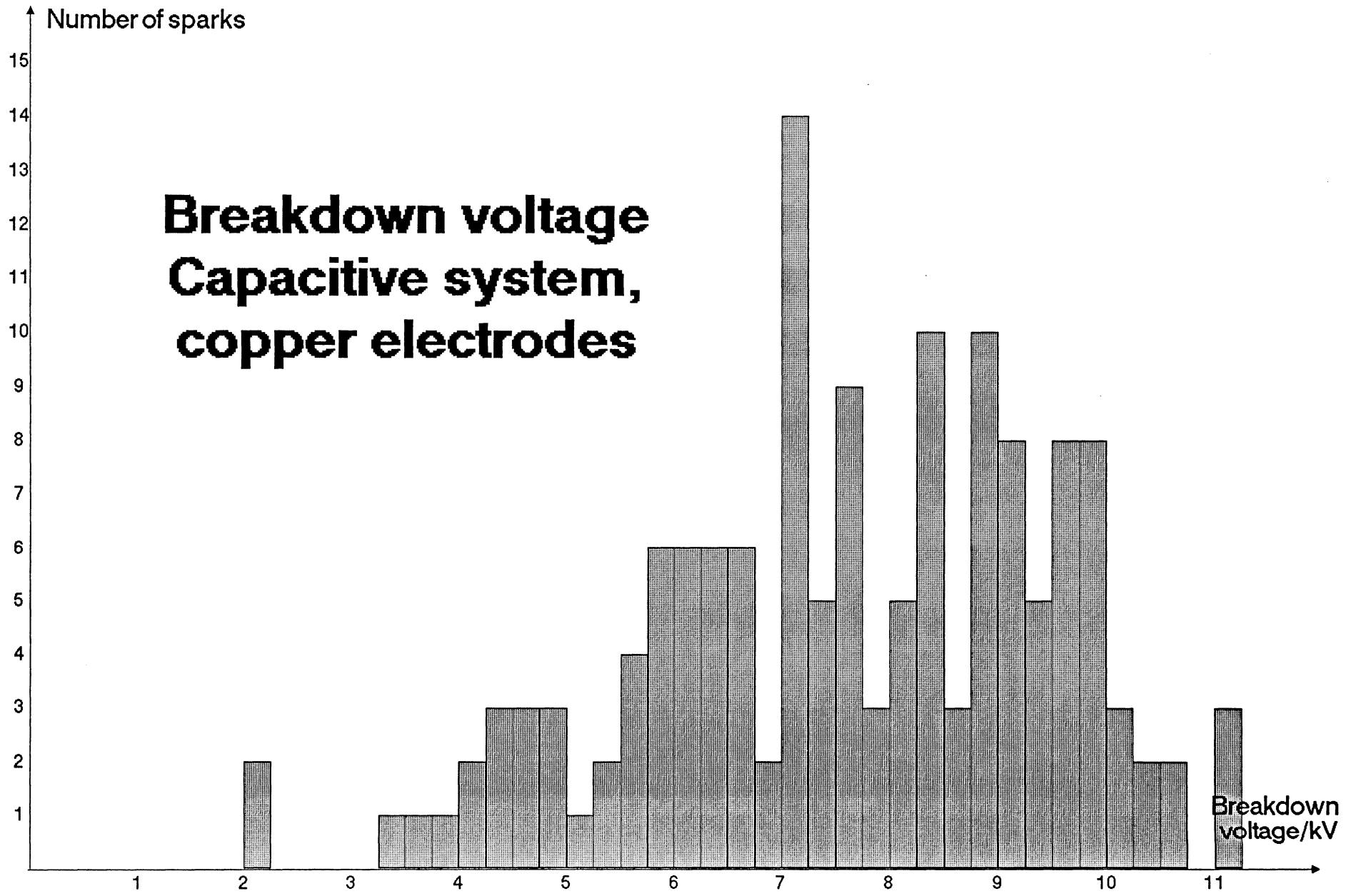


DIAGRAM 2.3.2 C

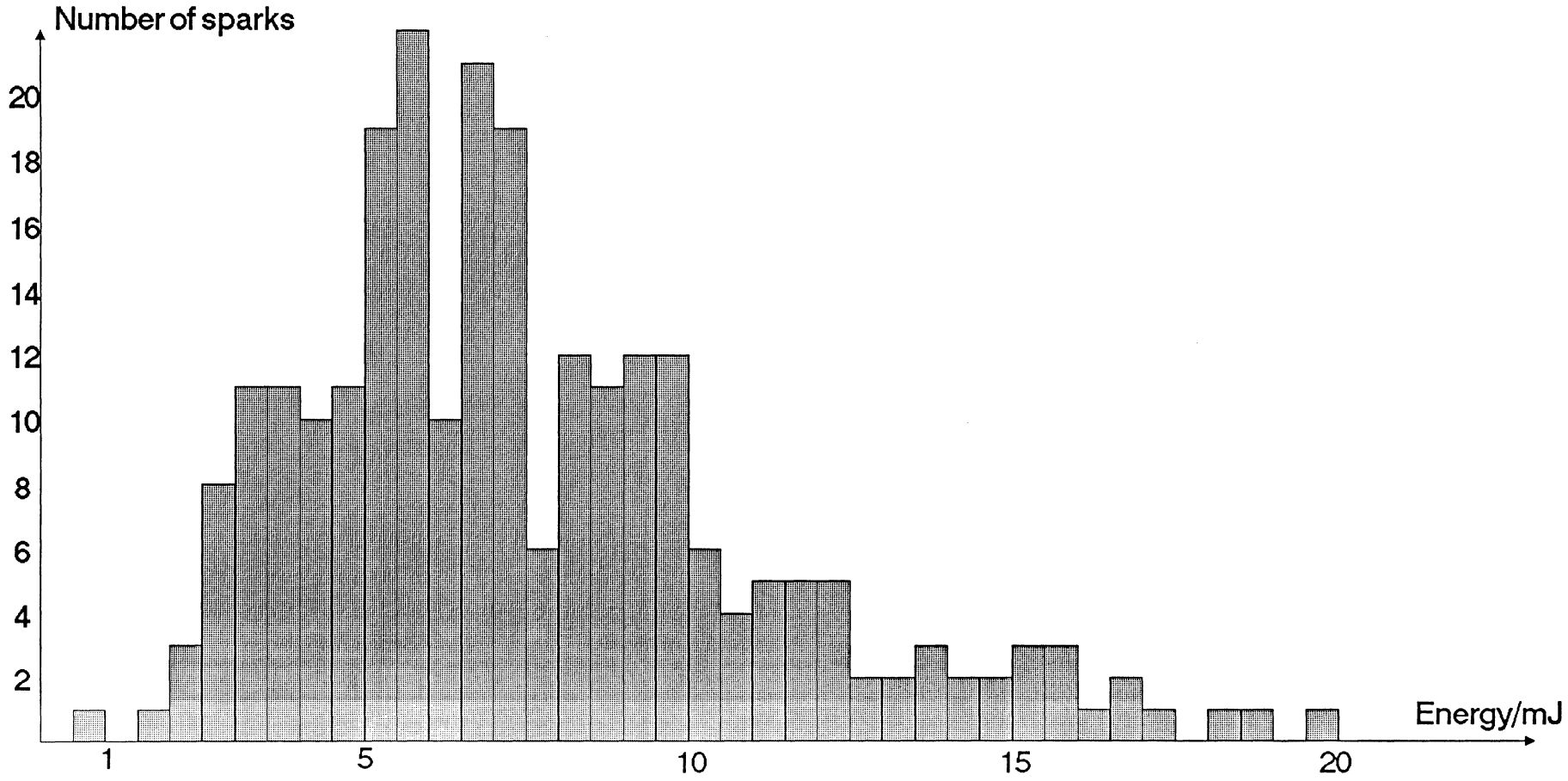


Breakdown voltage
Capacitive system, steel electrodes

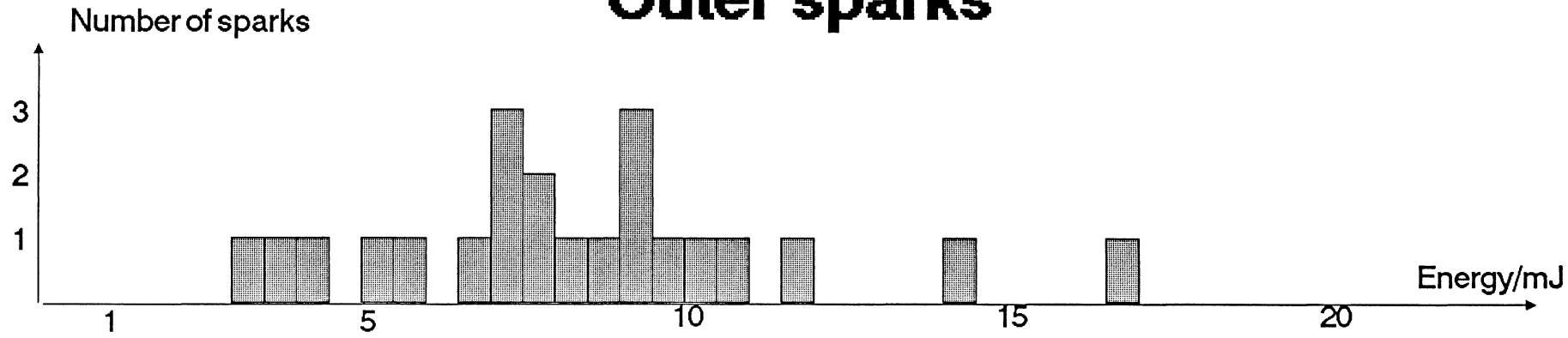



Number of sparks
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
Breakdown voltage
Capacitive system,
copper electrodes
1
2
3
4
5
6
7
8
9
10
11
Breakdown
voltage/kV

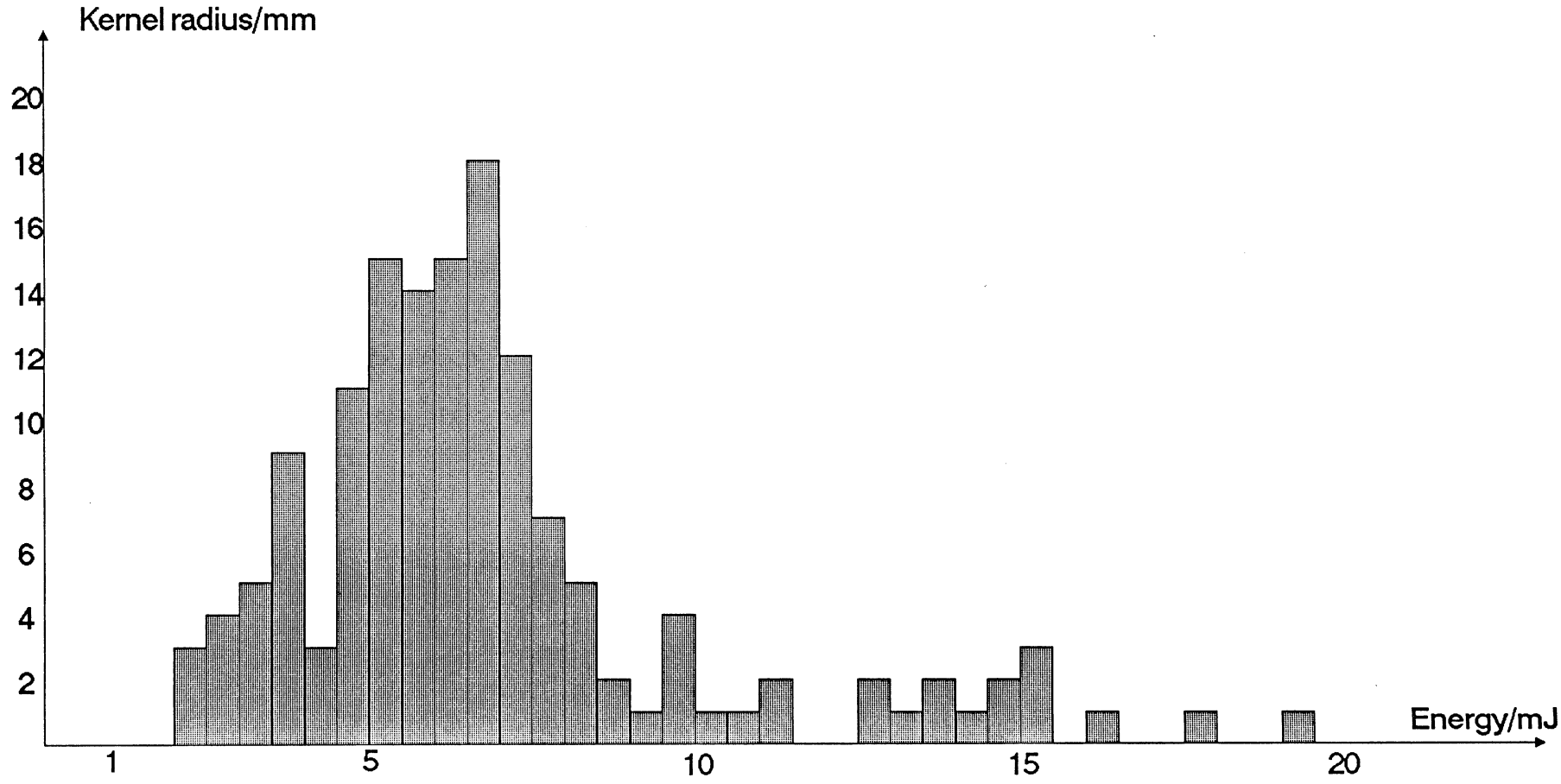
First part of energy
Inductive system, steel electrodes
250 sparks



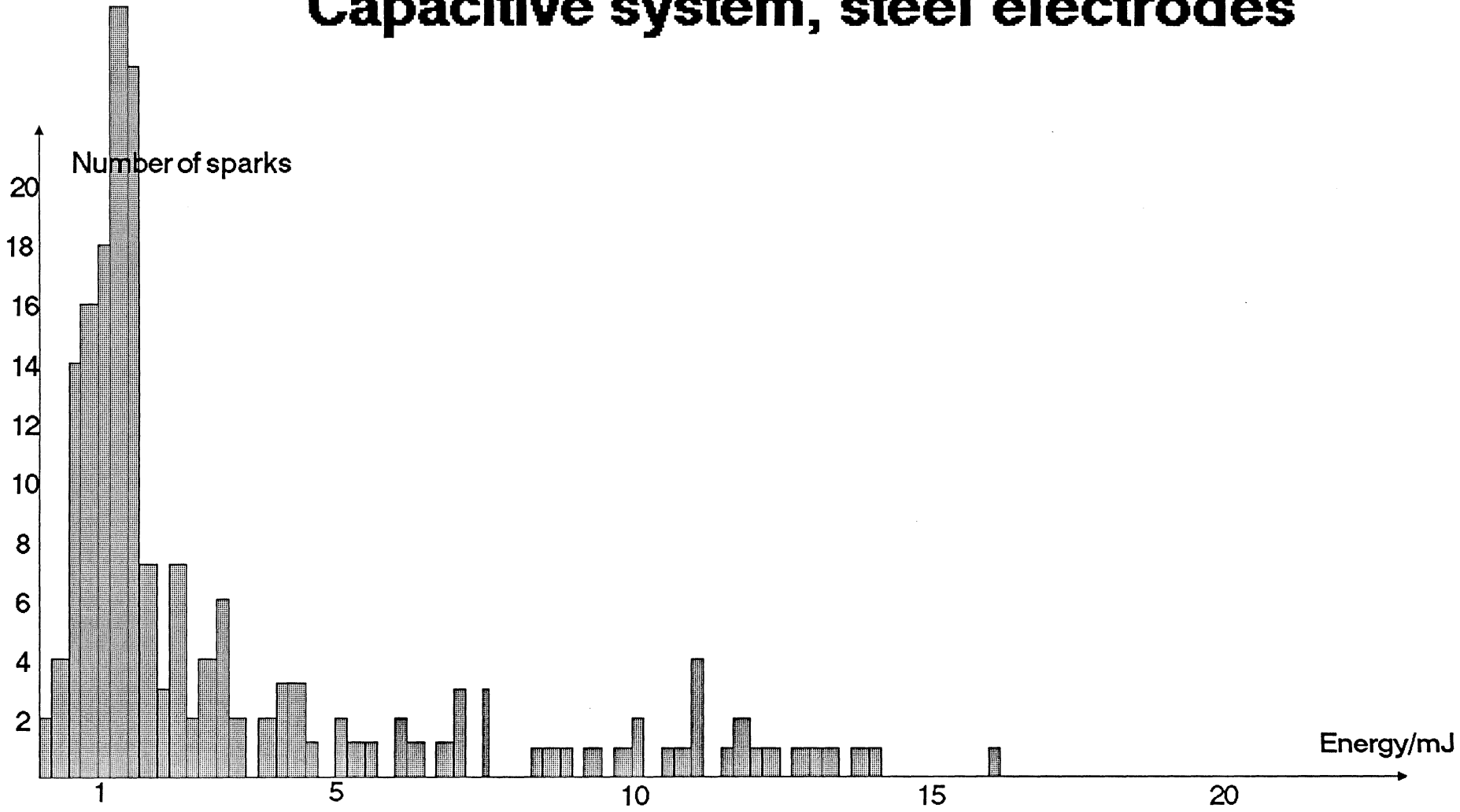
First part of energy
Inductive system, steel electrodes
Outer sparks



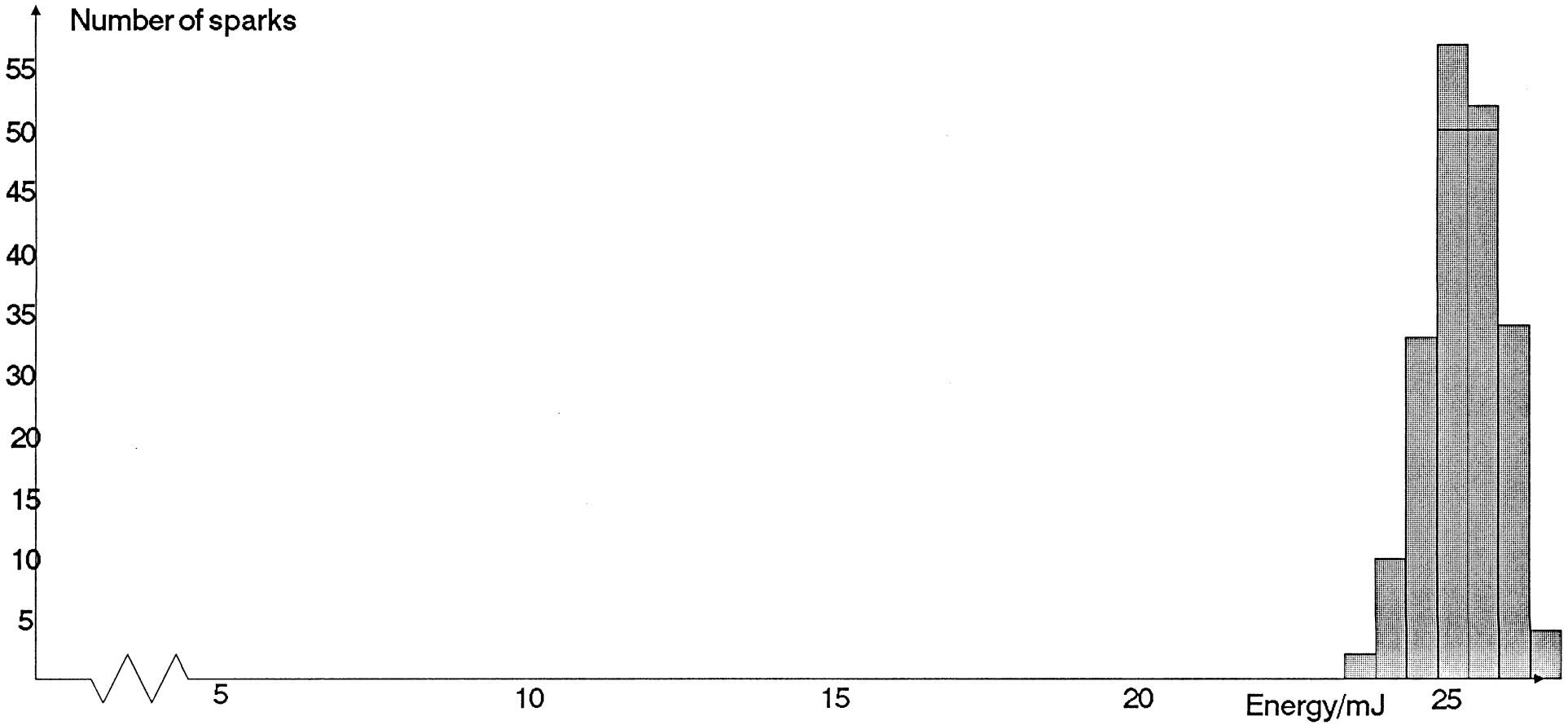
First part of energy
Inductive system, copper electrodes



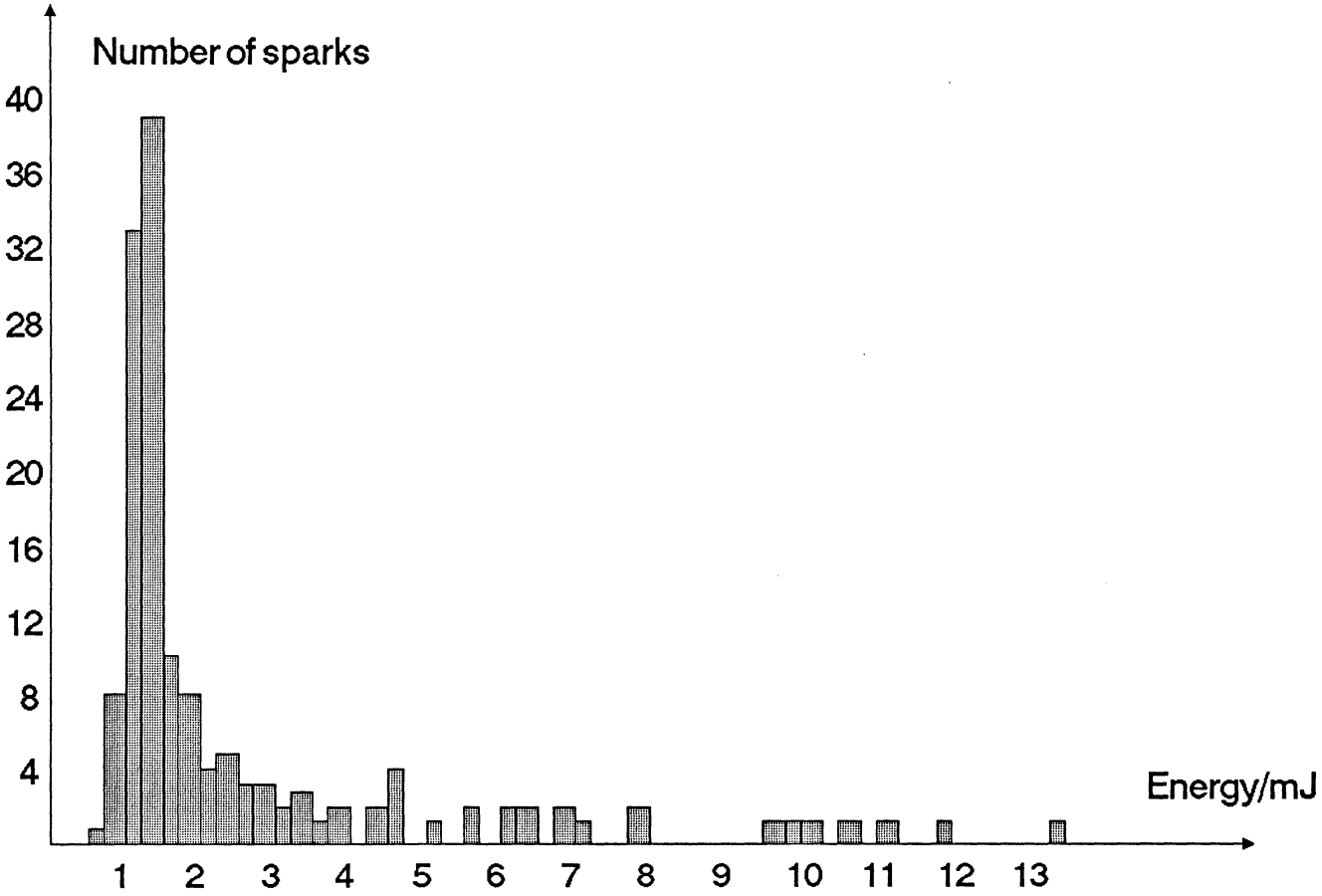
First part of energy
Capacitive system, steel electrodes



Second part of energy
Capacitive system, steel electrodes



First part of energy
Capacitive system, copper electrodes



Second part of energy
Capacitive system, copper electrodes

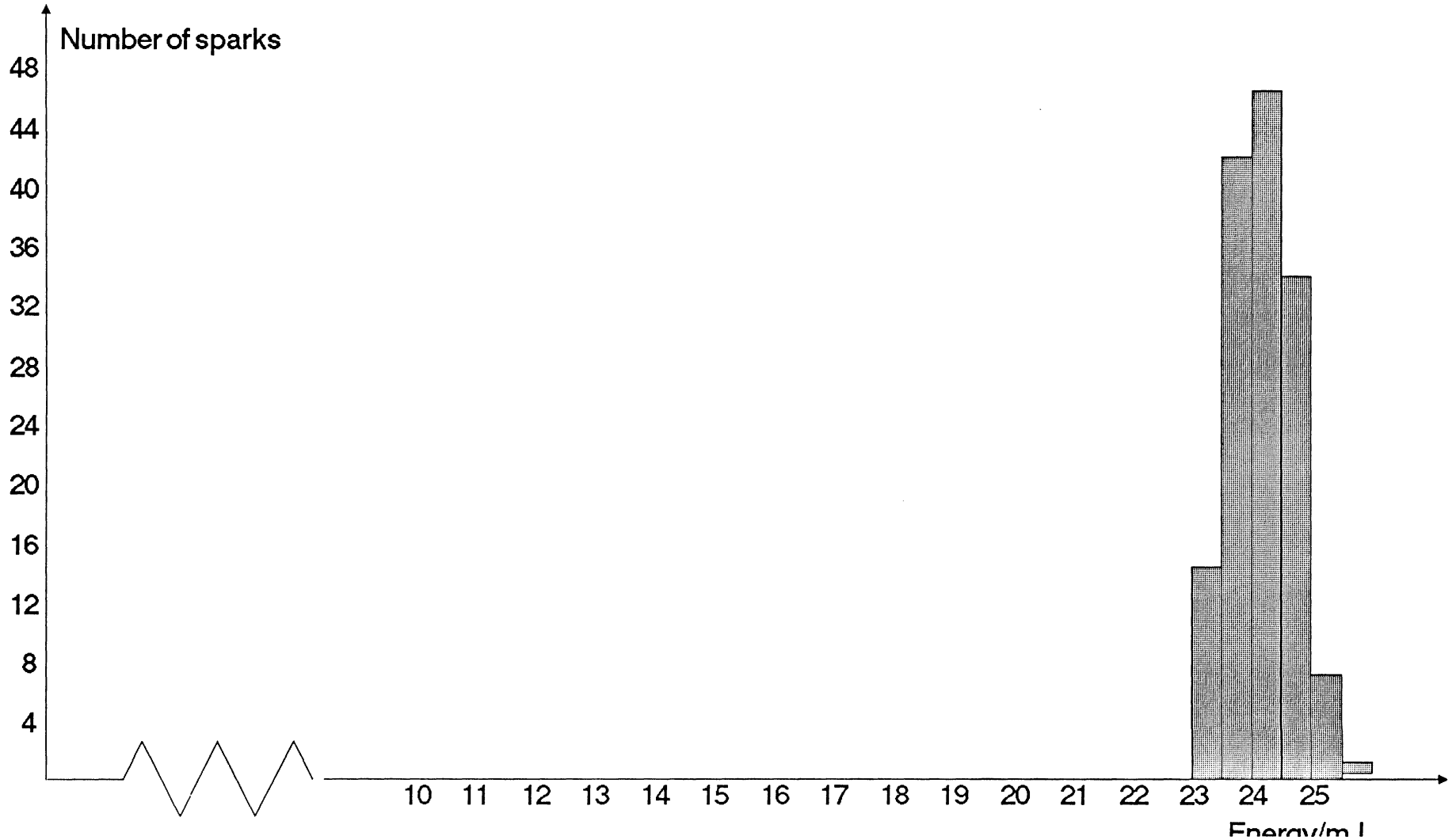
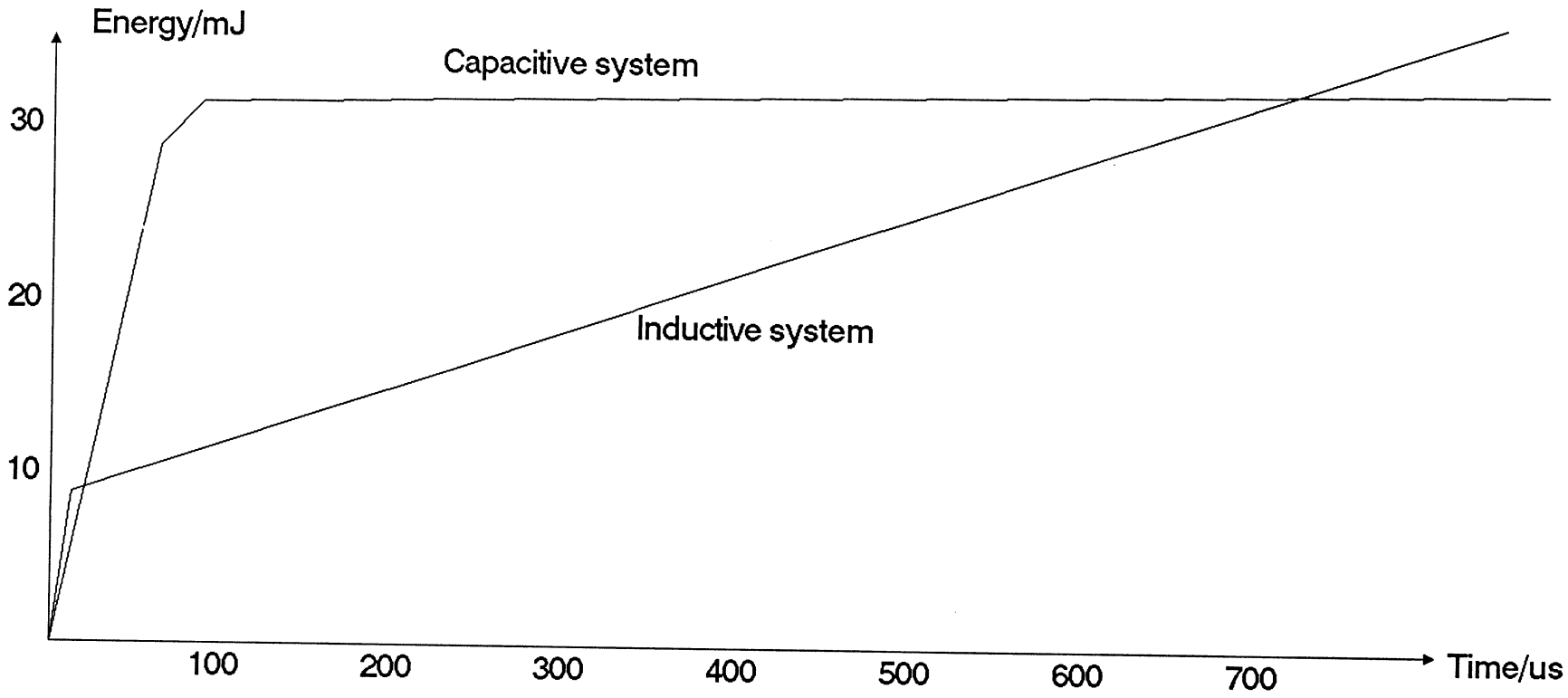
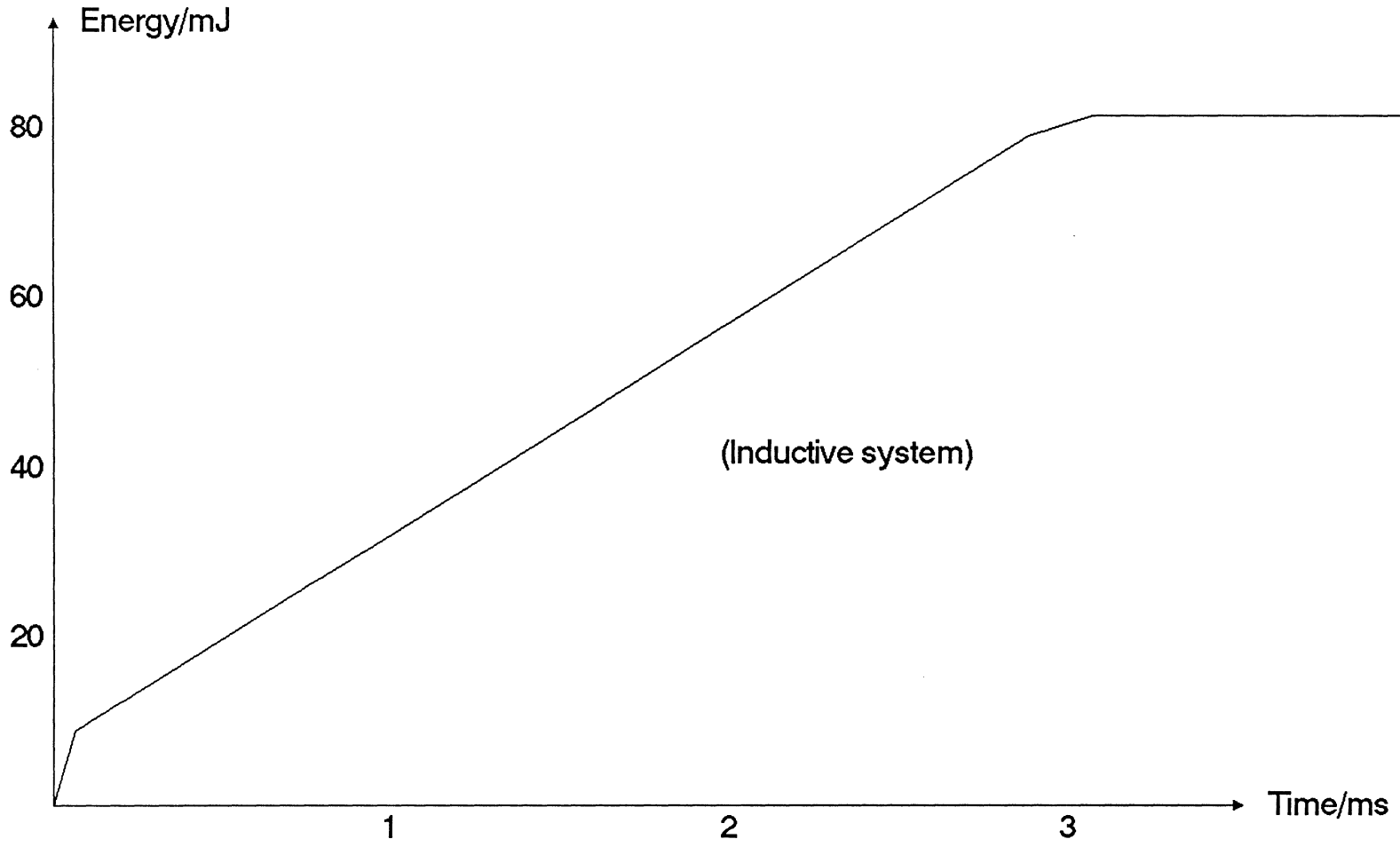
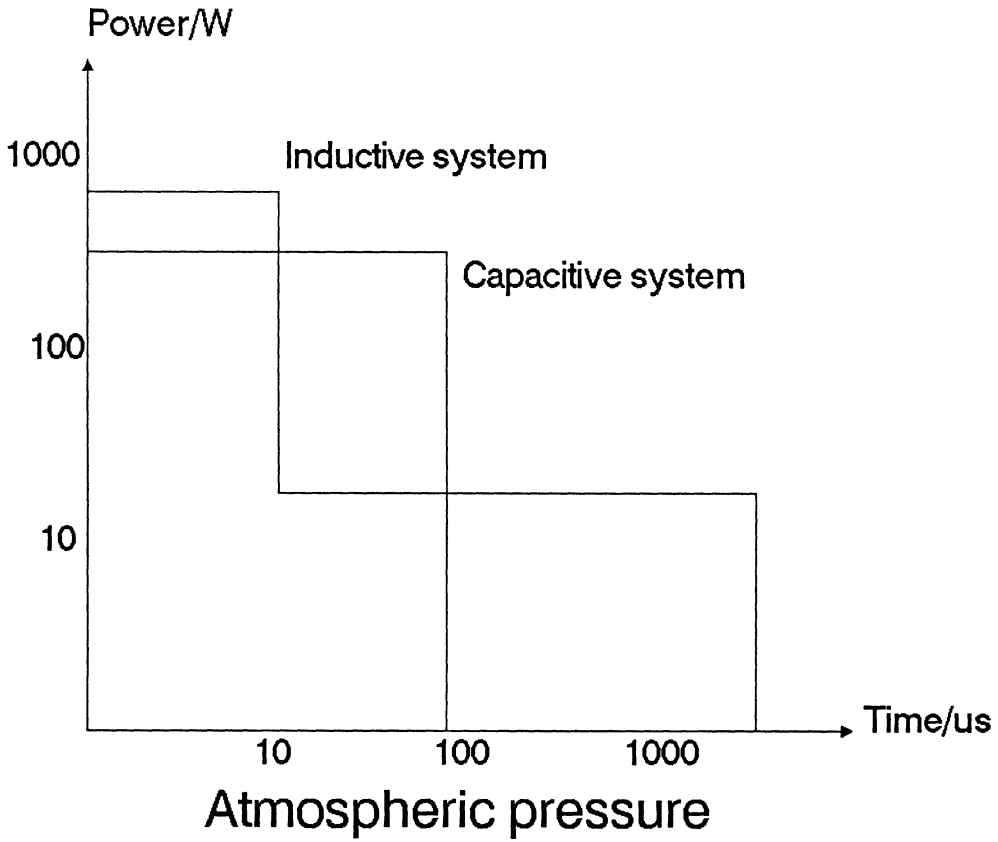


DIAGRAM 2.7


Energy/mJ
Capacitive system
30
20
Inductive system
10
100
200
300
400
500
600
700
Time/us


Energy/mJ
80
60
40
20
(Inductive system)
Time/ms
1
2
3


Power/W
1000
Inductive system
Capacitive system
100
10
Time/us
10
100
1000
Atmospheric pressure

2.4 Results

The results from the flame propagation measurements are shown in diagrams 2.3.1.A - E. From these diagrams one can see, in particular, two important characteristics. First, an extrapolation of the radius-versus-time line would not pass through the origin. Second, the speed of the flame front seems to alter abruptly at a radius of about 9-10 mm.

As the energy is primarily bound to the spark plasma, its transfer to the gas is of great importance. This transfer takes a relatively long time - from the breakdown there is no noticeable growth of the kernel for as long as almost one millisecond. During this time heat is transferred from the plasma to its surroundings, and ignition takes place. To optimize the heat transfer, the space angle occupied by the electrodes must be as small as possible. This is the case of the "outer sparks". Furthermore, the unwanted heating of the electrodes has very large cycle-to-cycle variations, which provides the same effect as a random delay of the spark itself. We believe this to be the most important factor causing variations in kernel radius at a certain time after breakdown.

The change in the flame front speed is of a more inevitable origin. In the beginning, the curvature of the front is not very different from the thickness of the front itself. Under such conditions, the flame front speed is considerably lower than normal. The values found in the litterature correspond well to the higher ones measured here. The dashed lines in diagrams 2.3.1 A, B, and E correspond to the theoretical value calculated for a propane/air mixture of 4.2% at room temperature, i.e. a progagation velocity of the flame front of about 2.95 m/s. We did not find any changes between copper and steel electrodes.

The distribution of the breakdown voltages is shown in diagrams 2.3.2. A - E. There is no correspondence between outer sparks and a high breakdown voltage (the distributions in diagrams 2.3.2 A and B are about

the same) which indicates that it is not a high breakdown voltage that generates an outer spark. The higher breakdown voltage and greater spread of the capacitive system are due to the much faster voltage ramp of the system. The effect of copper electrodes is not significant. The energy diagrams 2.3.3. A - G indicate no coupling between outer sparks and energy. The energies measured show extremely large variations, especially those associated with the capacitive system. Undoubtedly, the variations *are* large, but we strongly suspect electrical noise to have distorted the recorded current more than our digital filter could "straighten out". The filter (PROCEDURE SMOOTHCURRENT in the computer program, Appendix B) smoothens out all high-frequency disturbances, but cannot always give an alarm when receiving a "good-looking", but nevertheless, distorted current curve from the oscilloscope. However, one well-defined energy measured is the "second part" in the capacitive system, i.e. the energy dissipated 10-70 μs after breakdown. This energy shows practically no variation at all, as it originates from the oscillations of the capacitive discharge system and has nothing to do with the breakdown voltage. The diagrams from measurements with copper electrodes are almost identical.

As the impedance of the plasma does not depend on the breakdown voltage, the current, and hence the energy, show quite good correlation with the breakdown voltage. The statistical variations are large, as seen in diagrams 2.3.4 A and B, but nevertheless the dependence is clear.

Another dependence that one would intuitively search for is the one between energy and flame propagation. However all efforts here showed clearly that such a correlation did not exist.

3. High pressure measurements

3.1 Introduction

In order to more closly resemble conditions in a normal engine, we used a high-pressure cell with optical access through two quartz windows. These were ground with very high precision to approximately 5% of a wavelength; thus the interferograms would not be seriously affected. After ignition, the pressure in the cell rises by a factor of seven. Satisfactory operation of the valve was guaranteed only at pressures below 20 bar. In addition, the valve proved to "wait" for the 220 V A.C. to reach zero Volts, causing a delay evenly distributed between 0 and 10 ms. Considering this, we chose an initial pressure of 4 bar.

3.2 Experimental Details

The whole system for the gas supply had to be completely altered. Premixed air/fuel was stored in a gas tube. The fuel could be let into the cell via a series of valves after the cell had been evacuated. The gas system is shown in fig. 3.1.

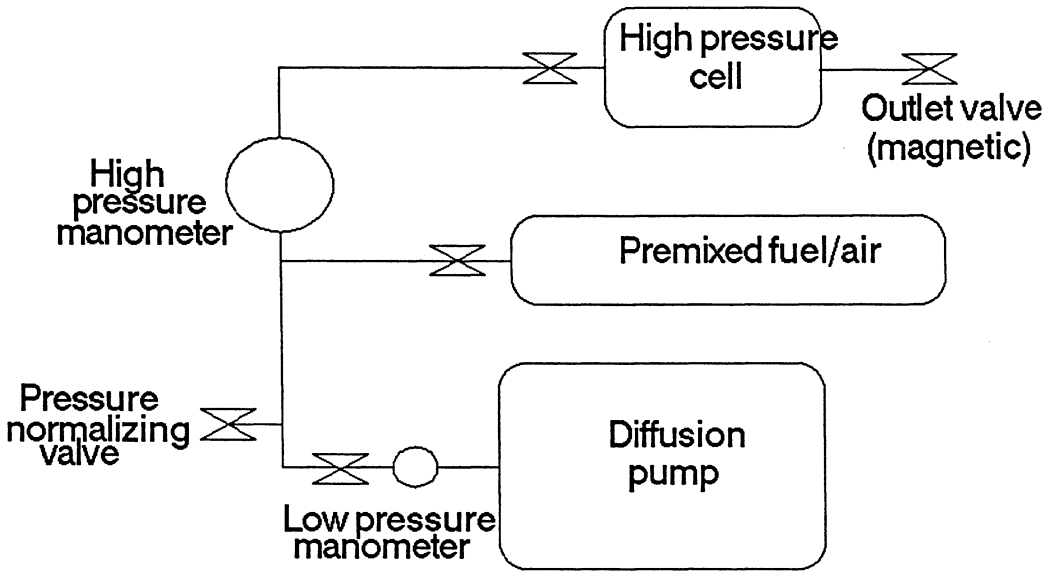


Fig. 3.1 The gas system

Electrically, however, many modifications had to be made before measurements could take place. The outlet valve - an ASCO B210B27 magnetic valve - had to be carefully synchronized. Opening too early would naturally have spoiled the measurement, and too late an opening (i.e. at a time when all or most of the gas in the cell had burned out) would have caused a dangerously high pressure in the cell. More disastrous was the fact that a commercial ignition system is designed to work at quite a high repetition frequency, and will not work at all below about 1 Hz. Furthermore, the very first spark given after a long non-active period ("long" is here to be interpreted as "more than a few seconds") will show characteristics very different from the desired ones. This problem was solved by two quite different approaches, one for each system. For the inductive system a device with three relays "evoked" the system and, after closing some switches, some 100 ms later gave another pulse, the latter one also synchronized with the pulses from the master trigger to laser and magnetic valve. For the capacitive system the solution was not so sophisticated, but nevertheless just as functional. Another spark plug, which gave a spark at a relatively low voltage, was connected to the first plug, and an extra trigger was used. Block schemes are shown in figs 3.2 - 3.3. At high pressure the ΔV values of fig. 2.6 become larger and the spark energy in diagram 2.7 and 2.8 is changed as shown in diagram 3.4 and the power level in diagram 2.9 is changed as in diagram 3.5.

Optically, not many changes were necessary. The quartz windows mentioned above caused no severe problems. After several ignition cycles, however, some vapour condensed on the inside of the windows and did not vaporize until after quite a long time - about five minutes. For future experiments, this delay will have to be decreased, for instance by letting pressurized air into the cell.

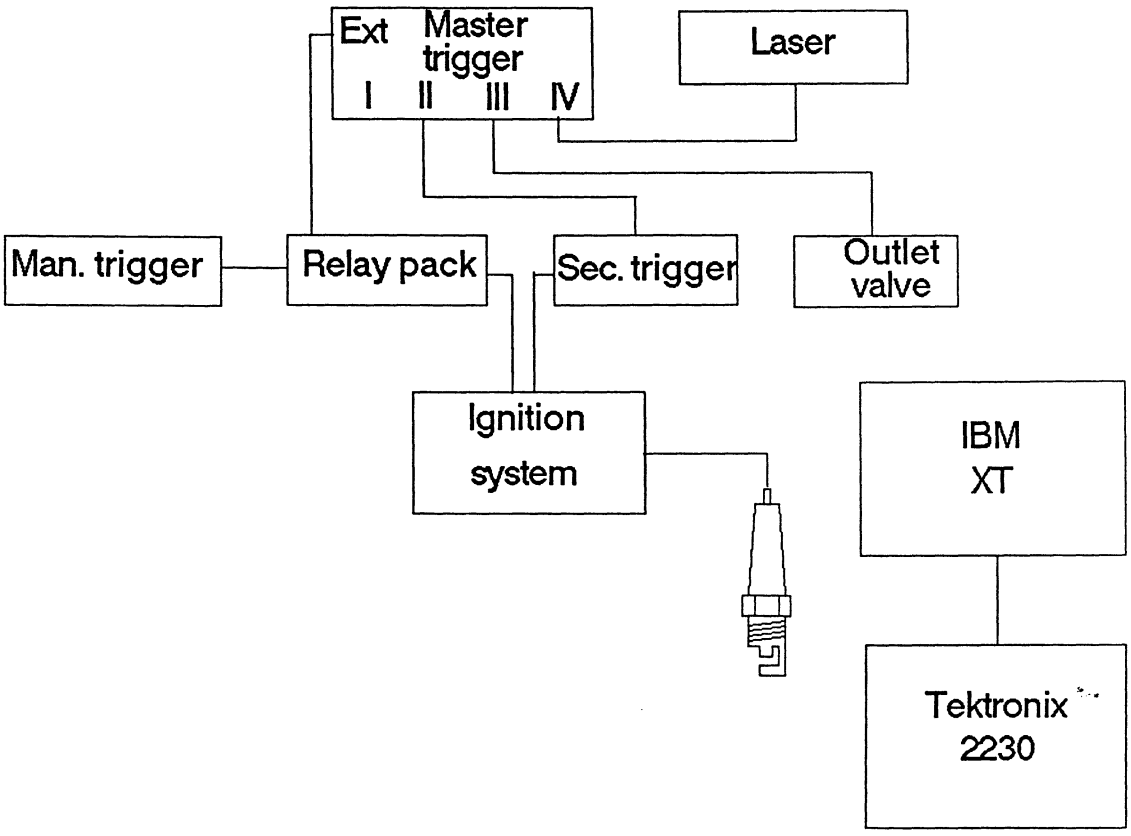


Fig. 3.2 Blockscheme inductive system

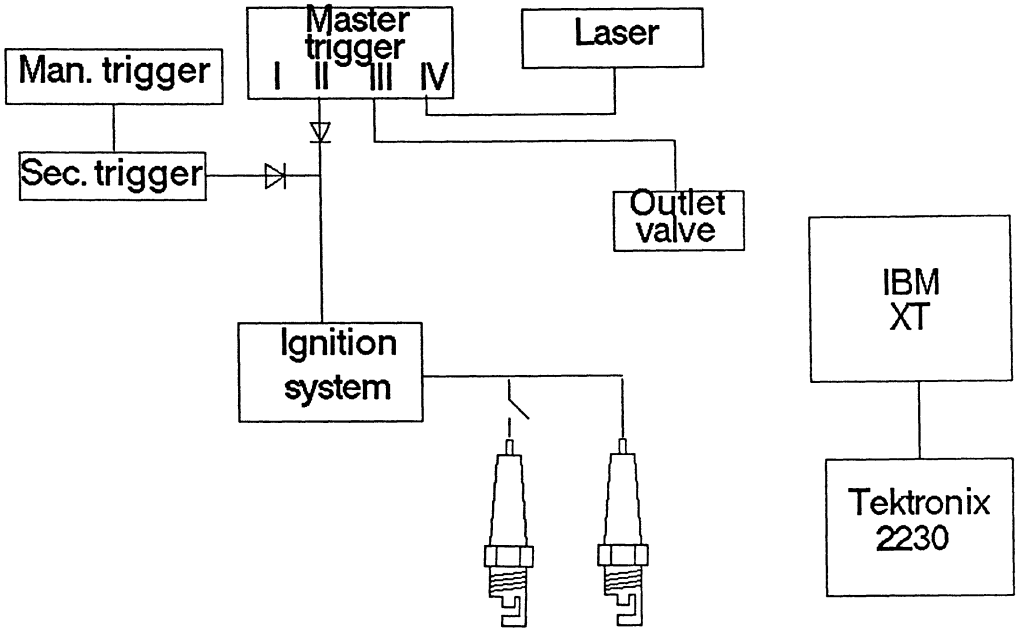


Fig. 3.3 Block scheme capacitive system

3.3 Diagrams

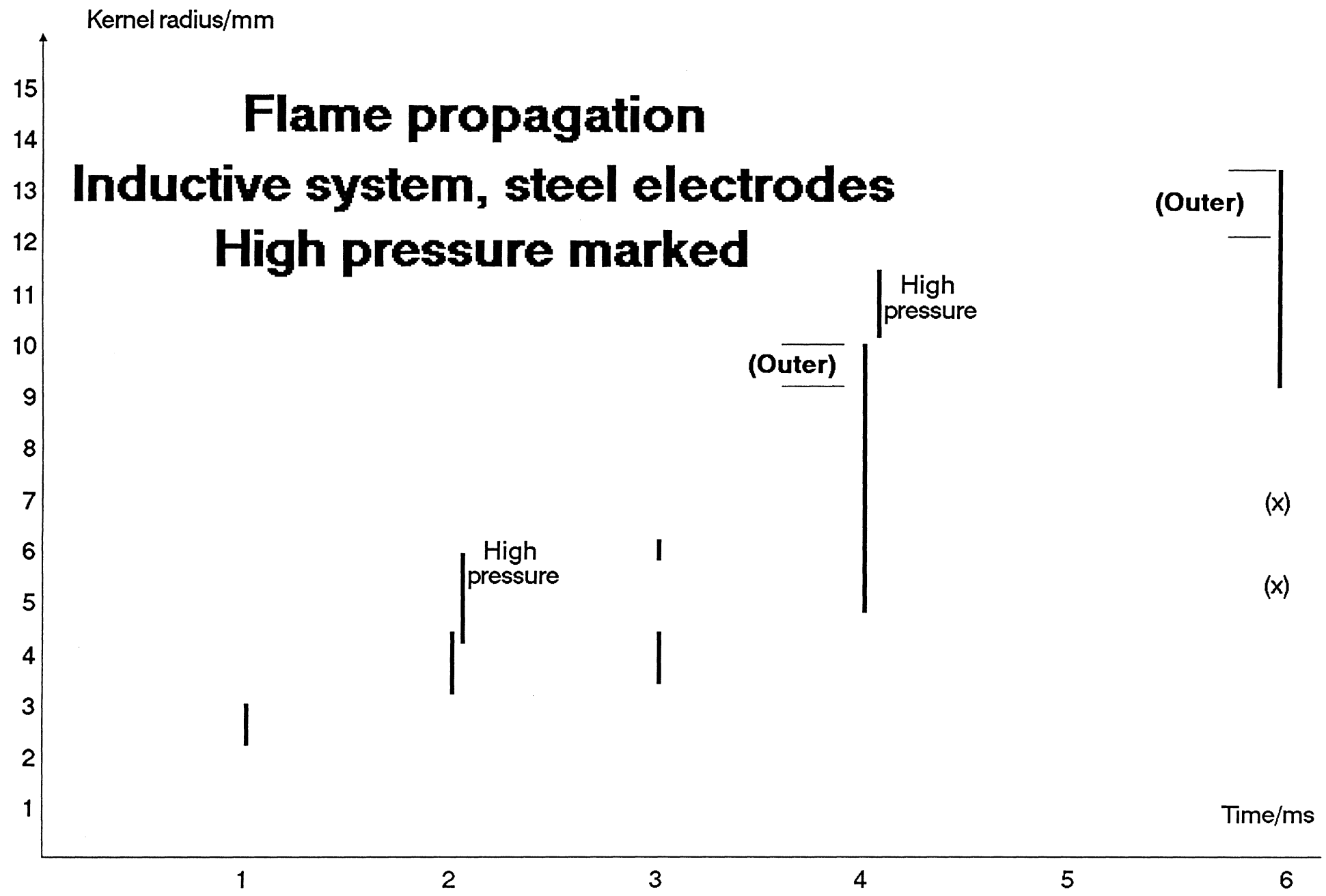
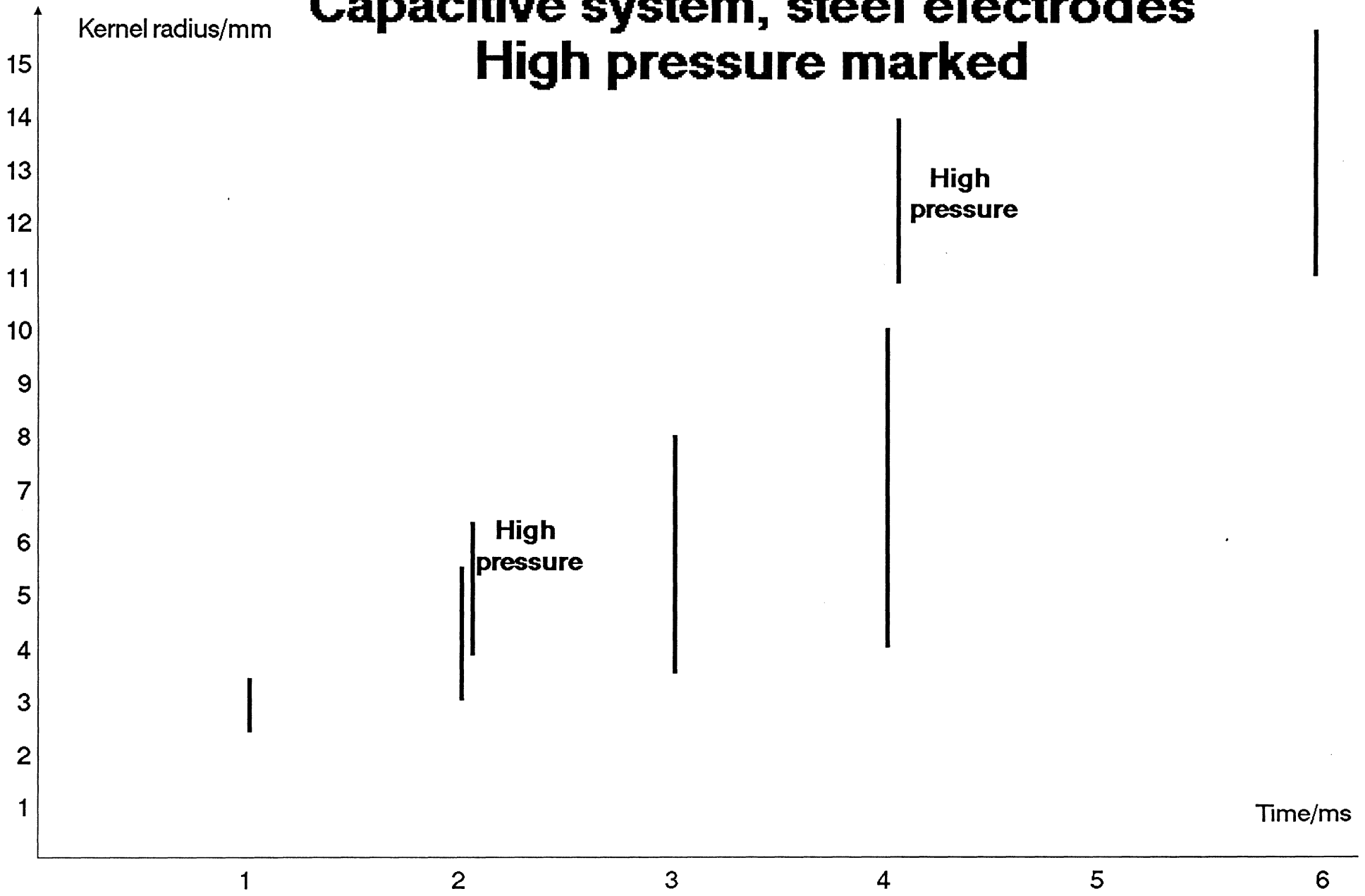

Kernel radius/mm
Flame propagation
Inductive system, steel electrodes
High pressure marked
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
(Outer)
High pressure
(Outer)
High pressure
(x)
(x)
Time/ms
1
2
3
4
5
6

DIAGRAM 3.3.1 B

Flame propagation
Capacitive system, steel electrodes
High pressure marked



Breakdown voltage
Inductive system, steel electrodes High pressure

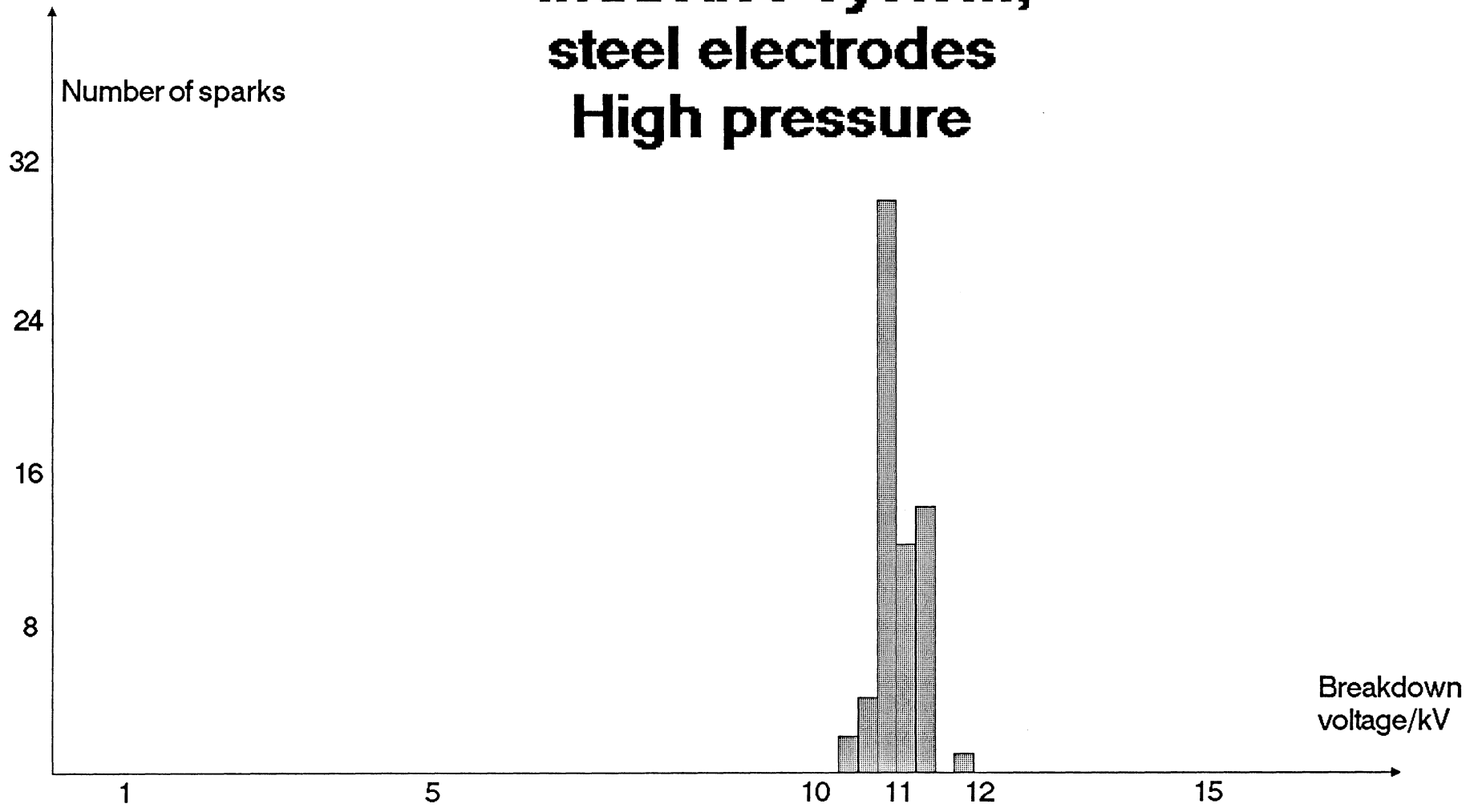
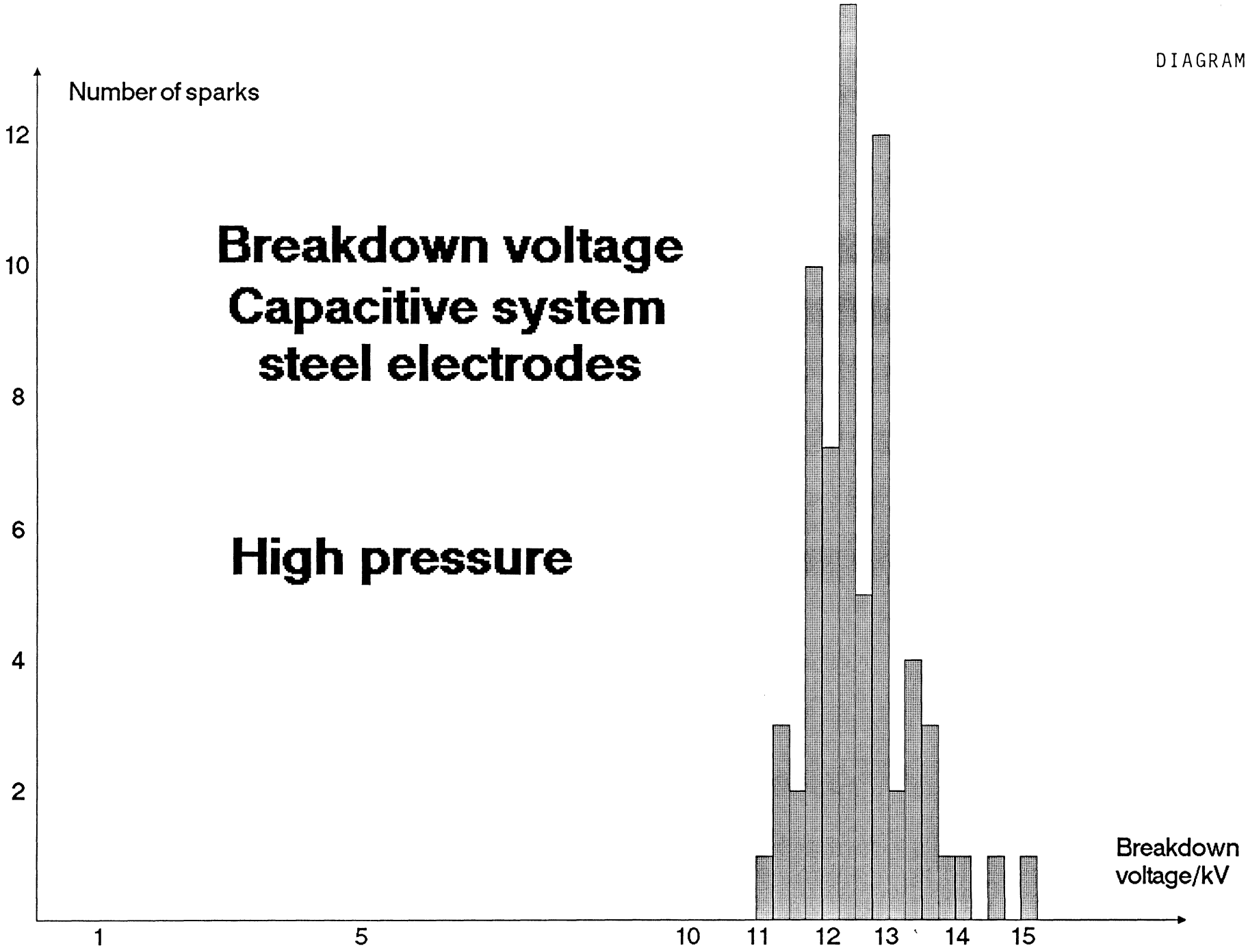
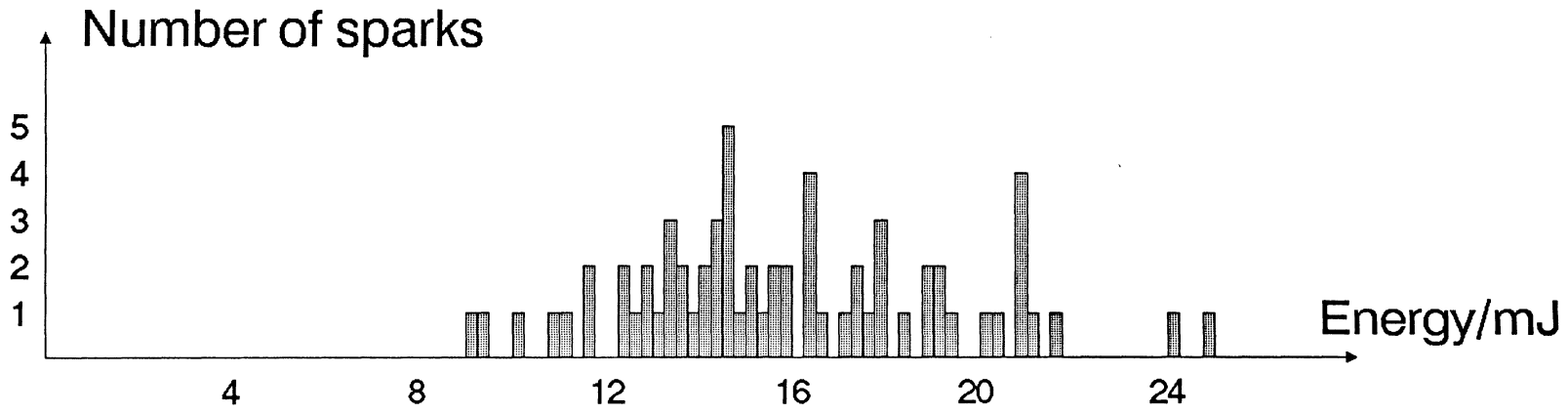


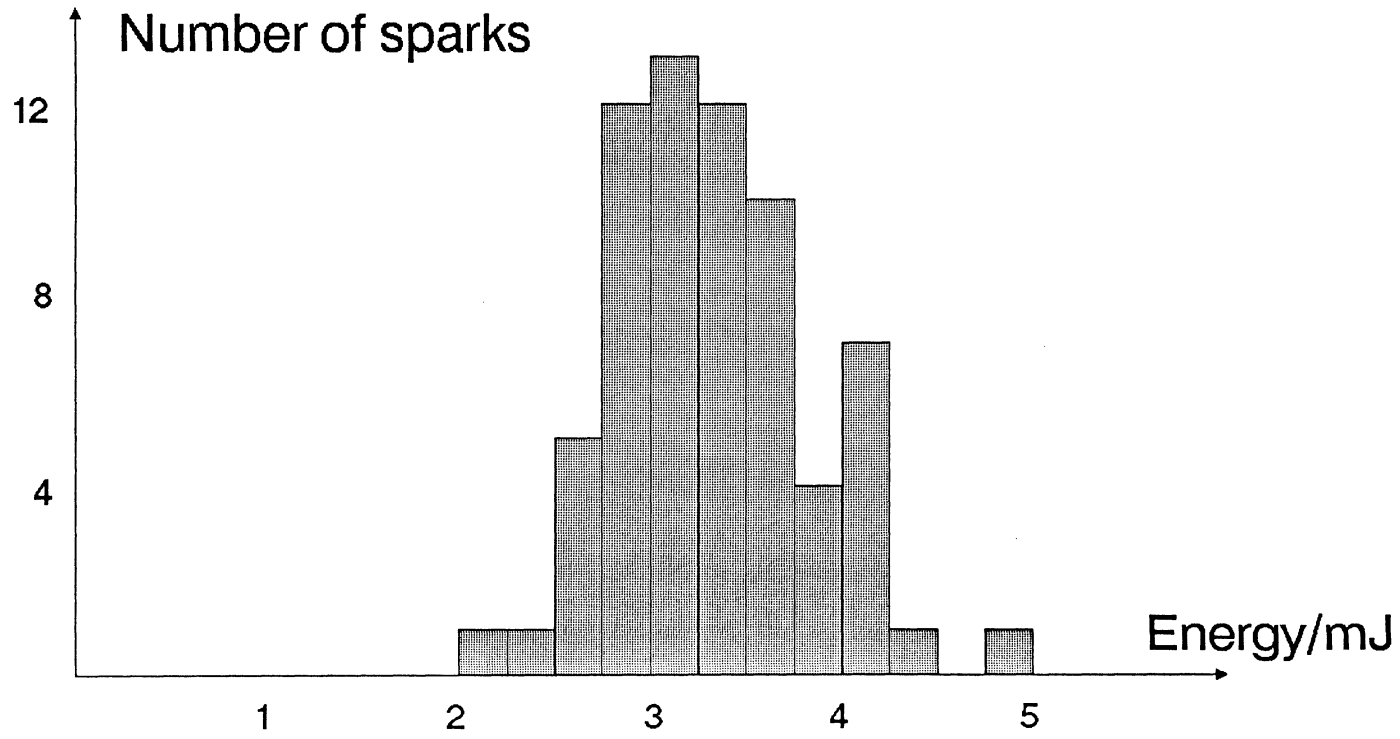
DIAGRAM 3.3.2 B



First part of energy
Inductive system, steel electrodes
High pressure



First part of energy
Capacitive system, steel electrodes
High pressure



Second part of energy
Capacitive system, steel electrodes
High pressure

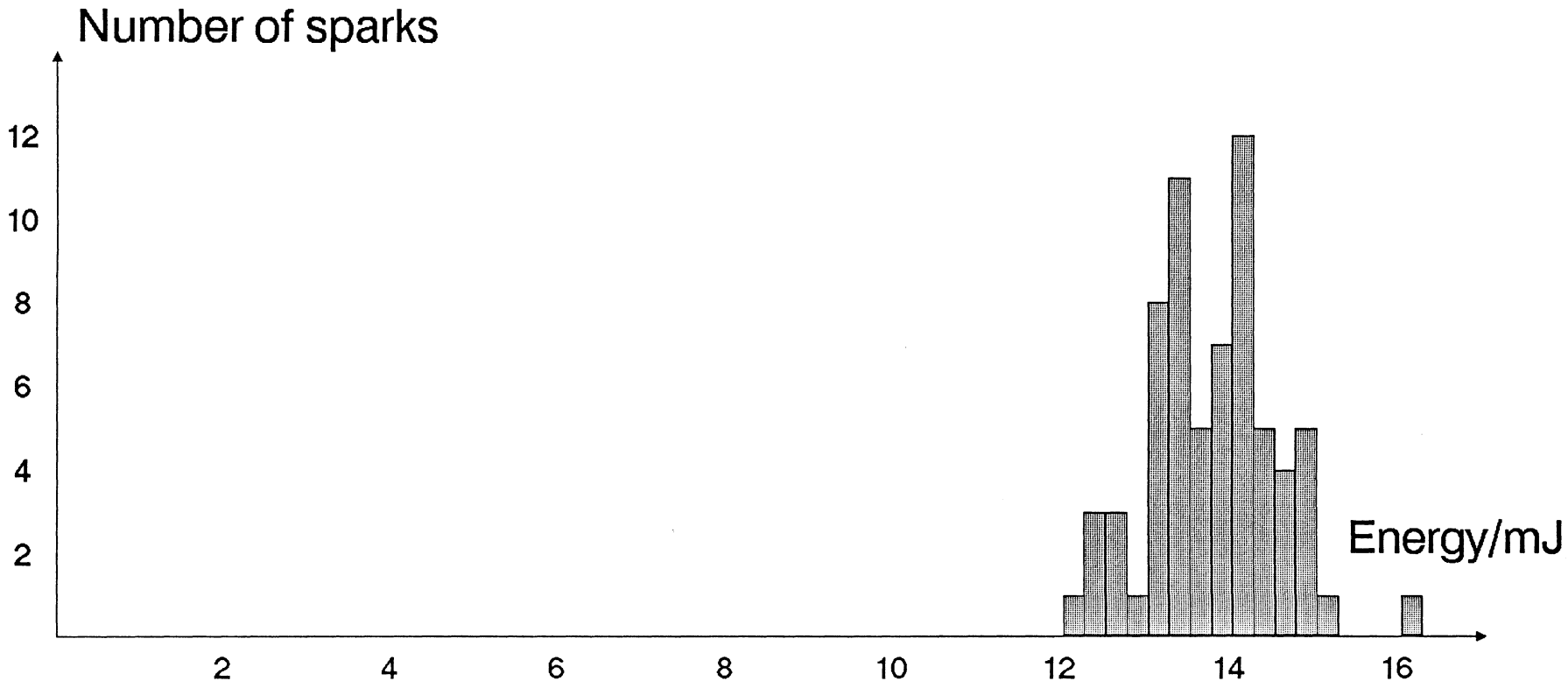
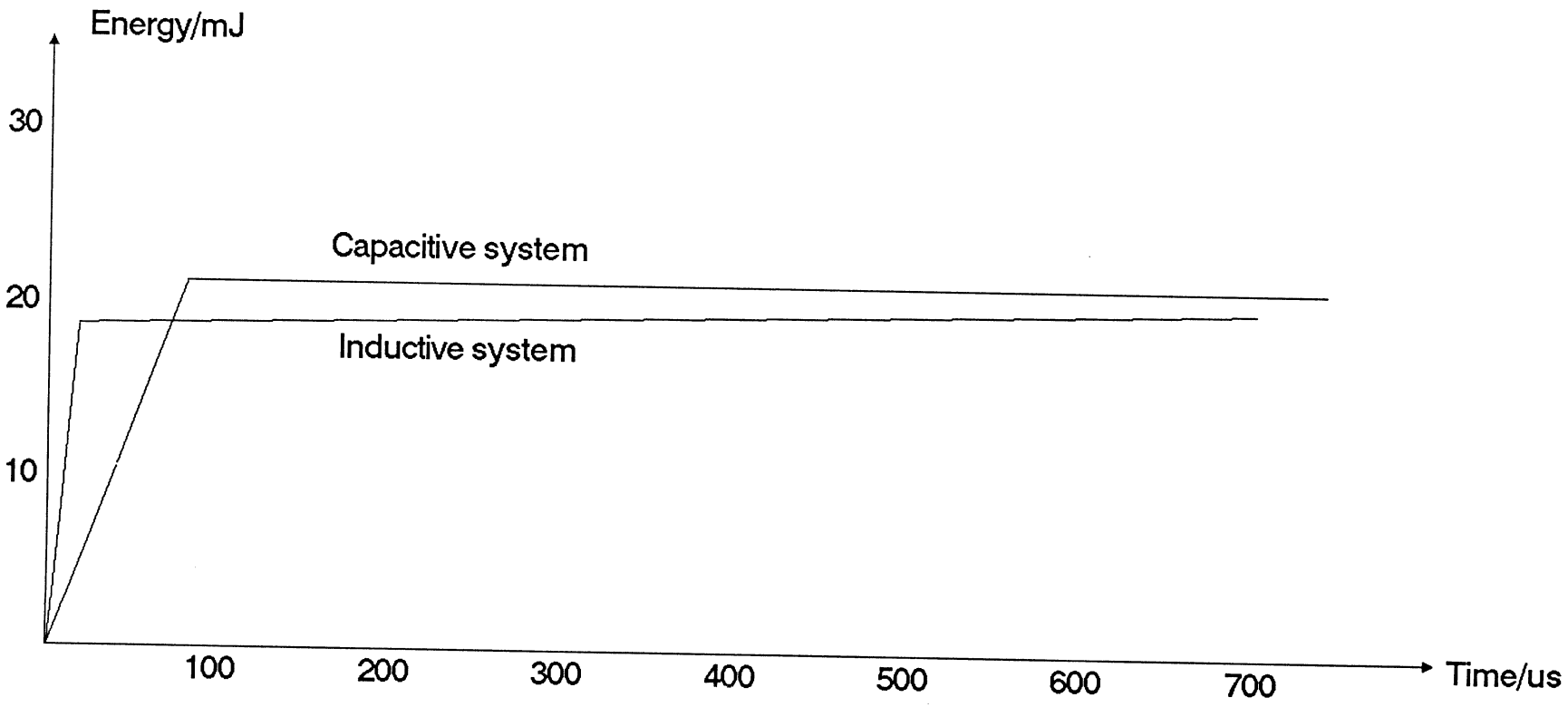
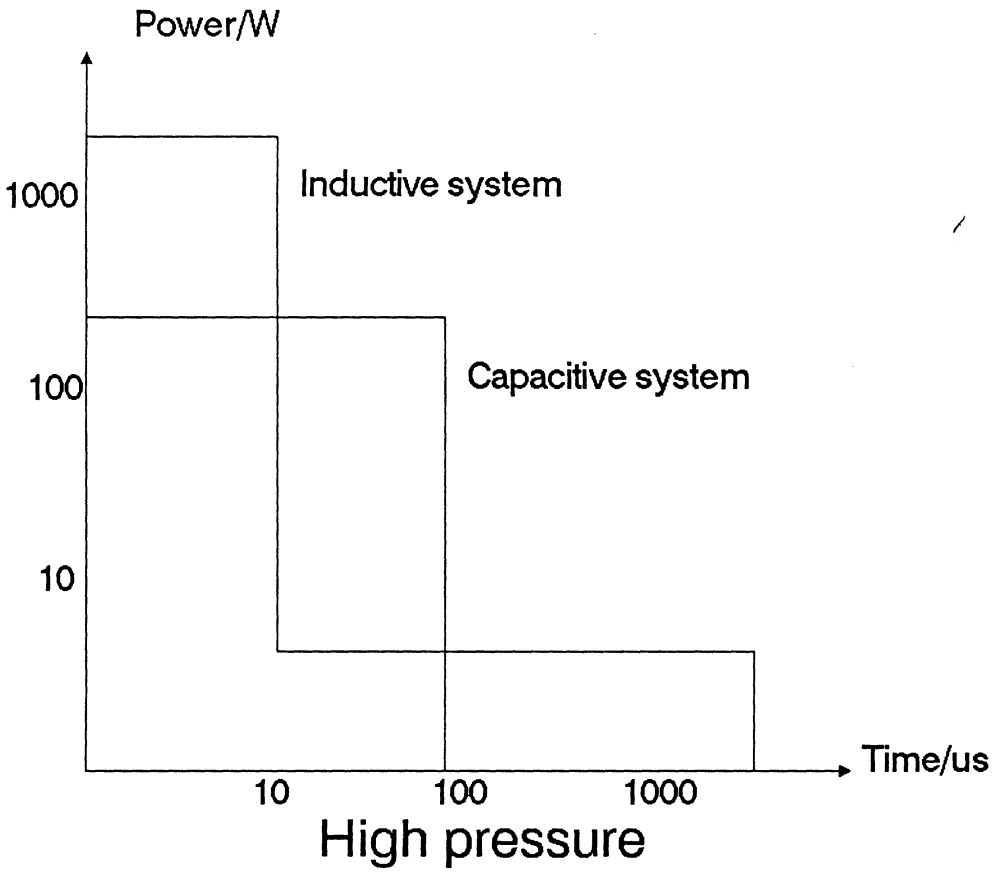


DIAGRAM 3.4


Energy/mJ
30
20
10
Capacitive system
Inductive system
100
200
300
400
500
600
700
Time/us


Power/W
1000
100
10
Inductive system
Capacitive system
Time/us
10
100
1000
High pressure

3.4 Results

The measurements at high pressure gave results very similar to the atmospheric ones. The results of flame propagation are shown in diagrams 3.3.1 A and B, the breakdown voltage distributions in diagrams 3.3.2 A and B, and the energy distributions in diagrams 3.3.3 A - C. The breakdown voltages, and consequently the energies, are greater, due to the better isolation of the compressed gas. An interesting result is that the spread in the breakdown voltage has become narrower, clearly for the capacitive system, but even more for the inductive one, probably because of its slower voltage ramp. The narrow spread does not depend solely on the pressure, although it seems to be the major factor. The premixing of air and fuel gives a very homogeneous gas mixture with no "lean spots" which could require abnormally high voltage if located at the electrodes. We could not find any correlations between the flame propagation and breakdown voltage or spark energy at high pressure.

Due to the small variations in energy and breakdown voltage, the correlations seen in diagrams 2.3.4 A and B of the atmospheric pressure measurement are not equally obvious at high pressure. As before, no correlation was found between energy and flame propagation.

4. Discussion

Cyclic variations originate during the initial stage of combustion, from the time of the breakdown to a "noticeable" departure of the cylinder pressure from the compression pressure [3, 4, 5]. A way of improving this early phase of combustion is through the ignition process. This is at least the method of most practical interest in the immediate future.

The total electrical energy dissipated per spark in a practical ignition system is between 30 and 100 mJ. There are several points of view, some direct contradictions, as to the best way of delivering this in terms of the nature and duration of the spark.

The spark of a typical coil ignition system consists of three different phases: the breakdown phase, the arc phase, and the glow phase, defined in different time intervals as shown in fig. 4.

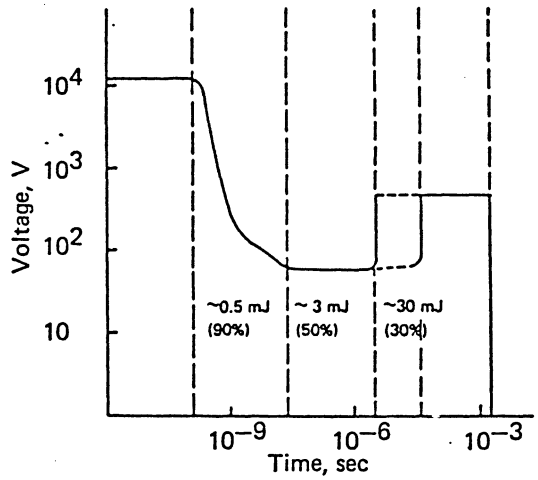


Fig. 4 The phases of the spark

Most of the energy is dissipated in the glow phase. According to Maly [13], these phases are not equally efficient in transferring heat to the gas. Comparing three different ignition systems, the breakdown, the capacitive discharge, and the inductive system, Maly argues the conversion efficiencies to be of the order of 90%, 50%, and less than 30% respectively. The measurements of Saggau [6] show the thermal conversion efficiencies to be of the order of 50%, 15%, and 10%. This is

due to the fact that the ignition system works at highly different power levels. In the first instance the spark must raise the temperature sufficiently in a small volume to cause thermal runaway. Then there is a certain minimum energy (depending on the gas, the pressure, the ignition system, etc.) needed before the flame has grown to a critical size and is able to propagate by itself as a combustion wave.

In the literature, there is no evidence that the amount of energy delivered is particularly critical as long as it exceeds the minimum energy (and we could not find any either). Increasing the pressure markedly decreases the minimum ignition energy [4, 43]. Since the ignition systems mentioned above work at different power levels, they also work at different time intervals as the total energy delivered is limited. Kono et al. [7] found that the optimum spark duration was between 50 μs and 300 μs, depending on the mixture strength and quenching action of the electrodes. With small electrode gaps, thick electrodes and lean mixtures, the optimum spark duration is longer. This means that energy dissipated long after breakdown does contribute to ignition, especially under difficult operating conditions when laminar burning velocity is low (i.e. low load, lean mixtures, high exhaust gas recirculation [8, 9, 10]). It is in these regions that cycle-to-cycle variation is at its worst. Longer arc duration and greater arc gaps improve combustion stability but the significance of the effect is dependent on ignition timing and engine load [11]. An increase in timing advance or a reduction in load (both of which reduce the temperature at the ignition time) will lead to an increased sensitivity to arc quality. There is a weak link between cycles and the magnitude of this link decreases with arc duration under conditions of large advance and low load.

In a real engine the fuel is unlikely to be completely premixed under all conditions at the time of ignition. Under such circumstances, if the discharge event were to last only a short time, there would be a possibility that it would take place when the local mixture strength was below the flammable limit. This could be alleviated, to an extent, by using wider spark gaps and longer spark duration (up to 5 ms) which provides a longer "time window".

However Maly has studied the discharge process in detail, and has argued

that the best results for ignition are obtained if the spark energy is concentrated in the breakdown component [12, 13, 14]. In an idealized engine with a homogeneous flow field and an accurate timing of the crank angle (spark advance) this might be more valid. This demands a breakdown that is well defined in time. This suggests that a capacitive discharge with its much faster voltage ramp, compared with an ordinary inductive system, would be of advantage.

Another way of improving the ignitability is through the geometry and material of the spark plug. Data in the literature indicate that larger electrodes increase heat transfer from the initial flame kernel and thereby increase the minimum ignition energy required. The rate of initial flame kernel development will also be adversely affected. This is in agreement with our measurements which showed that "outer sparks" propagate faster at early stages. Use of a plug with platinum tipped electrodes is proved to extend the lean ignition limit by allowing a wider gap and a reduced electrode diameter, which results in less heat loss from the initial flame kernel [3]. Another advantage of this plug is that it reduces electrode wear. In general, an increase in the spark gap improves ignitability. However, at the same time the voltage required for breakdown becomes higher, often exceeding the available voltage. Recent data prove the requirement of a larger spark gap with a capacitive discharge system in order to provide hydrocarbon emission comparable to that of an inductive system. On the other hand, the capacitive discharge system is capable of this due to its much faster voltage ramp.

Amongst the various approaches suggested, perhaps the best strategy is to increase the energy available to the gas throughout the period of early flame growth, i.e. while the flame front is still near enough to the spark discharge channel to be influenced by it. The lower the laminar burning velocity, the longer this period. With a short spark duration it becomes more important to ignite at a specific time as a long-duration spark provides a reduced sensitivity to the spark advance angle in engines.

5. Summary

Hier we are going to write a short summary.

6. Acknowledgements

We would like to thank the people who have given us help and support throughout our work. Göran Holmstedt at the Combustion Centre, Lund Institute of Technology, and Hans Johansson at Mecel AB, SAAB-Combitech, deserve special thanks. We would also like to thank Professor Thure Högberg at the Combustion Centre and Åke Bergquist at the Department of Atomic Physics.

7. References

1. Gautam T. Kalghatgi, Spark ignition, early flame development and cyclic variation in I.C. engines, SAE paper 870163.
2. Richard W. Anderson, The effect of ignition system power on fast burn engine combustion, SAE paper 870549.
3. Norihiko Nakamura, Tatsuo Kobayashi, Masanori Hanaoka, and Noboru Takagi, A new platinum tipped spark plug extends the lean misfire limit and useful life, SAE paper 830480.
4. Peters, B.D. and Borman, G.L., SAE paper 700064
5. Arrigoni, V., Calvi, F., Cornetti, G.M., and Pozzi, U., SAE paper 730088
6. Saggau, B., Calorimetry of the three discharge Modes of the electrical ignition spark, Archiv für Elektrotechnik, 64, 1981.
7. Kono, M., Humagai, S., and Sakai, T., Sixteenth symposium (international) on combustion, p. 757, The Combustion Institute, 1977.
8 Kono, M., Nakagawa, Y., Hamai, K., and Sonc, M., Stabilized combustion in a spark ignited engine through a long spark duration, SAE paper 850075.
9. Hancock, M.S., Buckingham, D.J., and Belmont, M.R., SAE paper 860321.
10. Hamai, K., Kawajiri, H., Ishizuka, T., and Nakai, M., Twenty-first symposium (international) on combustion, Munich 1986.
11. Hancock, M.S., Buckingham, D.J., and Belmont, M.R., The influences of arc parameters of combustion in a spark-ignition engine, SAE paper 860321.
12. Maly, R., and Vogel, M., Seventeenth symposium (international) on combustion, The Combustion Institute, 1979.
13. Maly, R., "Spark ignition: Its physics and effect on the internal combustion engine" in Fuel economy: Road vehicles powered by spark ignition engines, 1984.
14. Ziegler G.F.W., Wagner, E.P., Saggau, B., Maly, R., and Herden, W., SAE paper 840992.

Appendix A
The RS-232C interface

As an option, the Tektronix 2230 oscilloscope was equipped with an interface for RS232-C connection (option 12). The standard RS232-C connector consists of 7 pins, in general. However, for convenience, we chose to make a standard installation by using only three of them. The oscilloscope is "intelligent" enough to work with no more than three pins connected: Transmit, receive and signal ground (TXD, RXD, and GND). At the other end of the cable we had to "fool" the computer by connecting pins 5, 6, 8, and 20 to together. With this arrangement, the computer will interpret its own questions as answers to themselves. The effect of this is that all information will flow freely, as fast as possible. When in automatic mode, i.e. executing a program, our system still has to have some confirmation to be sure that everything sent is also received. Unfortunately, with no handshaking available we had to delay the program for some milliseconds to be sure of this. The loss in speed was, however, estimated to be less than 2%.

Appendix B
The calculation program

```
program inductive(input,output);

const       resistance          =  0.188; (* resistance of the current shunt *)
            arraylength         =  511;
            curvelength         =  2500;
            probeattenuation    =  1000;      (* For high voltage probe *)
            sparklimit          =  1;


type
   regpack             =  record
                            ax,bx,cx,dx,bp,si,di,ds,es,flags:integer;
                          end;
   charactervector     =  array [1..curvelength] of char;
   integervector       =  array [0..arraylength] of integer;
   realvector          =  array [0..arraylength] of real;
   posttype            =  record
                            nr        : integer;
                            deltat    : real;
                            deltav    : real;
                            firstpart : real;
                            total     : real;
                          end;


var     sampleinterval  : real;
        spark           : integer;
        i,j,k           : integer;
        channel         : integer;
        startindex,
        flankpos,
        flankindex      : integer;
        store           : charactervector;
        voltageint,
        currentint      : integervector;
        powerarray,
        energyarray     : realvector;
        resultfile      : file of posttype;
        OK              : boolean;
        badcurves       : integer;
        chan            : real;
        ch              : char;
        seconds,
        voltage         : real;

(*************************************************************************)
```

```
PROCEDURE SEND_CHAR(CH:CHAR);
var
   a:regpack;
begin
   a.dx:=0;
   a.ax:=$100+ord(ch);
   intr($14,a);
end;


(*****************************************************************************)

FUNCTION READ_CHAR:CHAR;
var
   a:regpack;
begin
   a.ax:=$300;
   a.dx:=0;
   intr($14,a);
   if (a.ax and 256 <> 0) then
   begin
      a.ax:=$200;
      a.dx:=0;
      intr($14,a);
      read_char := chr(lo(a.ax) and $7f);
   end else
      read_char:=CHR(0);
end;


(*****************************************************************************)


PROCEDURE SETCHANNEL(channel: integer);
begin
  case channel of
    1:  begin
          delay(1000);
          write(aux,chr(13));
          delay(100);
          write(aux,'CURSOR, CHANNEL:CH1',chr(13));
          delay(200);
          write(aux,'DATA, CHANNEL:CH1',chr(13));
          delay(200);
        end;
    2:  begin
          delay(1000);
          write(aux,chr(13));
          delay(100);
          write(aux,'CURSOR, CHANNEL:CH2',chr(13));
          delay(200);
          write(aux,'DATA, CHANNEL:CH2',chr(13));
          delay(200);
        end;
  end;
end;


(*****************************************************************************)


PROCEDURE READ_IN;
var   loop    : integer;
```

```
     ch     : char;
begin
 loop := 1;
 repeat
   repeat
     ch :=read_char;
   until ch<>CHR(0);
   store[loop] := ch;
   loop := succ(loop);
 until ((ch=chr(13)) and (loop > 5)) or (loop = curvelength);
 if loop = curvelength then
   writeln('Error while trying to read_in: No carriage return.');
end;

(*****************************************************************************)


PROCEDURE CURVEIN;
 /
var  k : integer;

begin
  write(aux,chr(13));
  delay(100);
  write(aux,'CURVE?',chr(13));
  READ_IN;
end;


(*****************************************************************************)


procedure TRANSFORM (var charvect : charactervector;
                     var intvect : integervector);


var    c, i      :  integer;
       tal       :  integer;
       zerolevel :  integer;

begin
  c := 1;
  i := 0;
  repeat
    while not (('0' <= charvect[c]) and (charvect[c] <= '9')) do
      c := c + 1;
    tal := 0;
    while ('0' <= charvect[c]) and (charvect[c] <= '9') do
    begin
      tal := 10*tal + ord(charvect[c]) - ord('0');
      c := c + 1;
    end;
    intvect[i] := tal;
    i := i + 1;
  until (charvect[c] = ';') or (c = curvelength);
  if c = curvelength then
    writeln('FEL I TRANSFORM!')
  else
    writeln('Curve transformed to array with ',i,' elements:');
  zerolevel := 0;
```

```
  for i := 1 to 50 do
    zerolevel := zerolevel + intvect[i];
  zerolevel := round(zerolevel/50);
  writeln('Zerolevel=',zerolevel);
  delay(2000);
  for i := 0 to arraylength do
  begin
    intvect[i] := intvect[i] - zerolevel;
  end;
end;



(*****************************************************************************)


PROCEDURE WAITFORSPARK;

var  j,k,a    : integer;

begin
  write(aux,chr(13));
  DELAY(100);
  write(aux,'SGLSWP ARM',chr(13));
  writeln('SINGLESWEEP ARMED.');
  j := 1;
  repeat
    delay(100);
    write(aux,'SGL?',chr(13));
    read_in;
    k := 1;
    repeat
      write(store[k]);
      k := k + 1;
    until ((store [k] = chr(13)) and (k>5)) or (k = 15);
    j := j + 1;
  until ((store[5] ='D') and (store[6] = '0')) or (j = 300);
  writeln;
  if j = 300 then                              (* j = 300 means time = 30 sec *)
  begin
    writeln('Oscilloskopet triggar inte.');
    OK := false;
  end;
end;


(*****************************************************************************)


FUNCTION READVALUE: REAL;

var    answer      :  array [1..30] of char;
       i,k,
       exponent    :  integer;
       ch          :  char;
       value,
       weight      :  real;

begin
  k := 1;
  exponent := 0;
  repeat
    repeat
```

```
    ch := read_char;
  until ch <> chr(0);
  answer[k] := ch;
  k := k + 1;
until (ch = chr(13)) or (k = 30);
writeln('All characters are read.');
writeln ('SENT FROM SCOPE   ');
I := 1;
repeat
  write(' ',ANSWER[I]);
  i := i + 1;
until answer [i] = chr(13);
k := 4;                                  (* Begin search at FIFTH position *)
repeat
  k := k + 1;
until (answer[k] in ['0'..'9']) or (k = 30);
if k <> 30 then
begin
  value := 0;
  while answer[k] in ['0'..'9'] do
  begin
    value := 10*value + ord(answer[k]) - ord('0');
    k := k + 1;
  end;
  if answer[k] = '.' then
  begin
    k := k + 1;
    weight := 1;
    while answer[k] in ['0'..'9'] do
    begin
      weight := 0.1*weight;
      value := value + weight*(ord(answer[k]) - ord('0'));
      k := k + 1;
    end;
  end;
  if answer[k] = 'E' then
  begin
    k := k + 1;
    if answer[k] = '-' then
      exponent := - (ord(answer[k+1]) - ord('0'))
    else if answer[k] = '+' then
      exponent := ord(answer[k+1]) - ord('0')
    else
      exponent := ord(answer[k]) - ord('0');
  end;
  value := value * exp(exponent * ln(10));
  readvalue := value;
end
else
  readvalue := 0.0;
end;


(*****************************************************************************)


PROCEDURE FINDPOSITIONS;

(* Note that when scope is in dual channel mode, the 1024/4096 points
are shared between the channels. Consequently, each 'integervector' will
contain 512/2048 integers, BUT the scope will nevertheless report posi-
tions from 2 to 1048/4096. No odd numbers are reported. Not to be confused,
carefully notice the use of 'index' numbers and 'position' numbers. *)
```

```
const  downinterval     =  10;
       upinterval       =  2;
       downsteps        =  10;
       upsteps          =  4;


var    newpoint,
       ready            :  boolean;
       index,
       steps            :  integer;
       i                :  integer;

begin
  index := 0;
  steps := 0;
  newpoint := false;
  ready := false;
  repeat
    repeat
      if voltageint[index+downinterval] < voltageint[index] then
      begin
        steps := steps + 1;
        index := index + downinterval;
      end
      else
      begin
        newpoint := true;
        index := index - steps*downinterval + 1;
        steps := 0;
      end;
    until (steps = downsteps) or (newpoint);
    if steps = downsteps then
    begin
      startindex := index - (downsteps-1)*downinterval - 1;
      write(aux,'CURSOR, SELECT:CURS1, TARGET:ACQ, POSITION:',2*startindex,chr(13));
      delay(100);
      ready := true;
    end;
  until (ready) or (index >= 400);
  OK := index < 400;
  if OK then
  begin
    writeln('OUT OF FIRST OUTER REPEAT. POS = ',2*startindex);


    steps := 0;
    newpoint := false;
    ready := false;
    index:=startindex;
    repeat
      repeat
        if voltageint[index+upinterval] > voltageint[index] then
        begin
          steps := steps + 1;
          index := index + upinterval;
        end
        else
        begin
          newpoint := true;
          index := index - steps*upinterval + 1;
          steps := 0;
        end;
      until (steps = upsteps) or (newpoint);
```

```
      if steps = upsteps then
      begin
        flankindex := index-(upsteps-1)*upinterval - 1;
        flankpos := flankindex * 2;
        write(aux,'CURSOR, SELECT:CURS2, TARGET:ACQ, POSITION:',flankpos,chr(13));
        delay(100);
        ready := true;
      end;
    until (ready) or (index >= 500);
    writeln('OUT OF SECOND OUTER REPEAT. POSITION = ',flankpos);
    OK := index < 500;
  end;
end;



(*************************************************************************)



PROCEDURE SMOOTHCURRENT(startindex,stopindex:integer);

const tolerance   =  20;

var   index,
      flipout,
      goodagain   :  integer;
      i, k        :  integer;
      oldvalue    :  integer;
      diff        :  real;

begin
  index := startindex;
  repeat
    repeat
      oldvalue := currentint[index];
      index := index + 1;
    until (abs(currentint[index] - oldvalue) > tolerance) or (index = stopindex);
    flipout := index;
    i := 0;
    if index <> stopindex then
      repeat
        oldvalue := currentint[index];
        index := index + 1;
        if abs(currentint[index] - oldvalue) <= tolerance then
          i := i + 1
        else
          i := 0;
      until (i = 3) or (index = stopindex) or (index - flipout >= 50);
    if i = 3 then
    begin
      goodagain := index - 3;
      diff := (currentint[goodagain] - currentint[(flipout-1)])/(goodagain-flipout+1);
      for k := flipout to (goodagain-1) do
        currentint[k] := currentint[(flipout-1)] + round(diff*(k-flipout+1));
    end;
  until (index >= stopindex) or (index - flipout >= 50);
  OK := (index >= stopindex);
  if not OK then
    for i := 100 to stopindex do
      write(currentint[i],' ');
end;
```

```
(*****************************************************************************)



PROCEDURE CALC;

var     i,j,k           : integer;
        voltdiv1,
        voltdiv2      : real;
        coeff         : real;
        trigpos,
        trigindex     : integer;
        timeincr      : real;
        thispost      : posttype;
        channel       : integer;
        dummy         : real;


begin
  write(aux,'CH1? VOLTS',CHR(13));
  voltdiv1 := readvalue;
  write(aux,'CH2? VOLTS',CHR(13));
  voltdiv2 := readvalue;
  coeff := (voltdiv1 * voltdiv2 * probeattenuation) / (25 * 25 * resistance);
  writeln ('Coefficient is ', coeff,'.');
  writeln ('Voltdiv1 = ',voltdiv1, ' Voltdiv2 = ',voltdiv2);
  k := 0;
  i := flankindex;
  for j := 1 to (flankindex - 1) do
    energyarray[j] := 0;

  repeat
    powerarray[i] := voltageint[i] * currentint[i] * coeff;
    if powerarray[i] < 0 then
    begin
      powerarray[i] := 0;
      k := k + 1;
    end;
    energyarray[i] := energyarray[i-1] + powerarray[i]*sampleinterval;
    i := i + 1;
  until i = arraylength;

  with thispost do
  begin
    nr := spark;
    deltat := seconds;
    deltav := voltage;
    firstpart := energyarray[flankindex+50]; (* One division after breakdown *)
    total := energyarray[i-1];
  end;

  writeln('Calculation of power aborted at i = ',i);
  write(resultfile,thispost);

  with thispost do
    writeln(nr:2,'  ',deltat:8,'  ',deltav:10,'  ',firstpart:10,' ',total:10);

  writeln('Number of negative powers after breakdown = ',k);
end;
```

```
(*****************************************************************************)


PROCEDURE INIT;

begin
  auxinptr := ofs(read_char);
  auxoutptr := ofs(send_char);
  assign(resultfile,'RESULT.DAT');
  rewrite(resultfile);
  for i := 1 to curvelength do
    store[i] := chr(0);
  badcurves := 0;
  write(aux,chr(13));
  delay(100);
  write(aux,'REMOTE ON',CHR(13));
  delay(100);
  write(aux,'DATA, SOURCE:ACQ,TARGET:REF1,ENCDG:ASCII',chr(13));
  delay(100);
  write(aux,'WFM ENCDG:ASCII',chr(13));
  repeat
    delay(40);
    write(aux,'WFM? XINCR',chr(13));
    sampleinterval := readvalue;
    writeln('Sampleinterval is read to ',sampleinterval);
  until (sampleinterval > 0.0) and (sampleinterval < 1.0);
end;


(*****************************************************************************)

(*************** S T A R T  O F  M A I N  P R O G R A M ***************)

(*****************************************************************************)


begin
  init;
  spark := 0;
  repeat
    spark := spark + 1;
    OK := true;
(*    waitforspark;  *)
    writeln(#7);                    (* Beep *)
    writeln(#7);
    writeln('Armera skopet. Ge sedan ny gnista och tryck ENTER.');
    readln(ch);
    if OK then
    for channel := 1 to 2 do
    begin
      setchannel(channel);
      write(aux,'DATA? CHA',CHR(13));
      READ_IN;
      k := 1;
      repeat
        write(store[k],' ');
        k := k + 1;
      until (store[k] = chr(13)) or (k = 30);
      curvein;
      case channel of
        1 : begin
              transform(store, voltageint);
```

```
            OK := true;
            findpositions;
            delay(400);
            if not OK then
            begin
              writeln('The flanks of the voltagecurve are not found');
              badcurves := badcurves + 1;
            end;
            write(aux,'DELTAT?,VALUE',chr(13));
            seconds:=readvalue;
            write(aux,'DELTAT?, VALUE',CHR(13));
            seconds := readvalue;
            writeln('TIME IS READ TO ',seconds);
            write(aux,'DELTAV?, VALUE',CHR(13));
            voltage := readvalue*probeattenuation;
            writeln('VOLTAGE IS READ TO ',voltage);
          end;
      2 : if OK then
          begin
            transform (store, currentint);
            smoothcurrent(1,arraylength);
            if OK then
              calc
            else
            begin
              badcurves := badcurves + 1;
              writeln('Curve too noisy for acceptance!');
            end;
          end;
    end;
  end;

  write(chr(7));       (* Beep *)

  i := flankindex - 3;
  begin
    writeln('Push return.');
    readln(ch);
    for k := 1 to 20 do
    begin
      if i = flankindex then write('*');
      writeln(i:3,voltageint[i]:6,currentint[i]:9,powerarray[i]:12:2,energyarray[i]*100(
      i := i + 1;
    end;
  end;
  writeln('Number of generated sparks so far = ',spark);
until (spark - badcurves) >= sparklimit;
writeln('PROGRAM TERMINATED.');
writeln('NUMBER OF ACCEPTED CURVES     = ', spark - badcurves:3);
writeln('NUMBER OF NOT ACCEPTED CURVES = ', badcurves:3);
end.
```

```
   program capacitive (input,output);

const       resistance          =  0.188; (* resistance of the current shunt *)
            arraylength         =  511;
            curvelength         =  2500;
            probeattenuation    =  1000;      (* For high voltage probe *)
            offset              =  100;     (* With parallell spark plugs *)
            sparklimit          =  37;


type
   regpack             =  record
                            ax,bx,cx,dx,bp,si,di,ds,es,flags:integer;
                          end;
   charactervector     =  array [1..curvelength] of char;
   integervector       =  array [0..arraylength] of integer;
   realvector          =  array [0..arraylength] of real;
   posttype            =  record
                            nr        : integer;
                            deltat    : real;
                            deltav    : real;
                            firstpart : real;
                            total     : real;
                          end;


var     sampleinterval  : real;
        spark           : integer;
        i,j,k           : integer;
        channel         : integer;
        startindex,
        flankpos,
        flankindex      : integer;
        store           : charactervector;
        voltageint,
        currentint      : integervector;
        powerarray,
        energyarray     : realvector;
        resultfile      : file of posttype;
        OK              : boolean;
        badcurves       : integer;
        chan            : real;
        ch              : char;
        seconds,
        voltage         : real;



(*****************************************************************************)


PROCEDURE SEND_CHAR(CH:CHAR);
var
   a:regpack;
begin
   a.dx:=0;
   a.ax:=$100+ord(ch);
   intr($14,a);
end;


(*****************************************************************************)
```

```
FUNCTION READ_CHAR:CHAR;
var
   a:regpack;
begin
   a.ax:=$300;
   a.dx:=0;
   intr($14,a);
   if (a.ax and 256 <> 0) then
   begin
      a.ax:=$200;
      a.dx:=0;
      intr($14,a);
      read_char := chr(lo(a.ax) and $7f);
   end else
      read_char:=CHR(0);
end;


(*****************************************************************************)


PROCEDURE SETCHANNEL(c : integer);
begin
  case channel of
    1:  begin
          delay(1000);
          write(aux,chr(13));
          delay(100);
          write(aux,'CURSOR, CHANNEL:CH1',chr(13));
          delay(200);
          write(aux,'DATA, CHANNEL:CH1',chr(13));
          delay(200);
        end;
    2:  begin
          delay(1000);
          write(aux,chr(13));
          delay(100);
          write(aux,'CURSOR, CHANNEL:CH2',chr(13));
          delay(200);
          write(aux,'DATA, CHANNEL:CH2',chr(13));
          delay(200);
       end;
  end;
end;


(*****************************************************************************)


PROCEDURE READ_IN;
VAR LOOP   :INTEGER;
    CH     :CHAR;
 BEGIN
  LOOP :=1;
  REPEAT
    repeat
      ch :=read_char;
    until ch<>CHR(0);
    STORE[LOOP] :=CH;
    LOOP :=SUCC(LOOP);
  UNTIL ((CH=CHR(13)) and (LOOP > 5)) or (LOOP = curvelength);
  if loop = curvelength then
    writeln('Error while trying to read_in: No carriage return.');
```

```
END;

(*****************************************************************************)


PROCEDURE CURVEIN;

var  k : integer;

begin
  write(aux,chr(13));
  delay(100);
  write(aux,'CURVE?',chr(13));
  READ_IN;
end;


(*****************************************************************************)



procedure TRANSFORM (var charvect : charactervector;
                     var intvect : integervector);


var    c, i      :  integer;
       tal       :  integer;
       zerolevel :  integer;

begin
  c := 1;
  i := 0;
  repeat
    while not (('0' <= charvect[c]) and (charvect[c] <= '9')) do
      c := c + 1;
    tal := 0;
    while ('0' <= charvect[c]) and (charvect[c] <= '9') do
    begin
      tal := 10*tal + ord(charvect[c]) - ord('0');
      c := c + 1;
    end;
    intvect[i] := tal;
    i := i + 1;
  until (charvect[c] = ';') or (c = curvelength);
  if c = curvelength then
    writeln('FEL I TRANSFORM!')
  else
    writeln('Curve transformed to array with ',i,' elements:');
  zerolevel := 0;
  for i := 1 to 50 do
    zerolevel := zerolevel + intvect[i];
  zerolevel := round(zerolevel/50);
  writeln('Zerolevel=',zerolevel);
  delay(2000);
  for i := 0 to arraylength do
  begin
    intvect[i] := intvect[i] - zerolevel;
  end;
end;
```

```
(*****************************************************************************)

PROCEDURE WAITFORSPARK;

var  j,k,a    : integer;

begin
  write(aux,chr(13));
  DELAY(100);
  write(aux,'SGLSWP ARM',chr(13));
  WRITELN('SINGLESWEEP ARMED.');
  j := 1;
  repeat
    delay(100);
    write(aux,'SGL?',chr(13));
    read_in;
    k := 1;
    repeat
      write(store[k]);
      k := k + 1;
    until ((store [k] = chr(13)) and (k>5)) or (k = 15);
    j := j + 1;
  until ((store[5] ='D') and (store[6] = '0')) or (j = 300);
  writeln;
  if j = 300 then                              (* j = 300 means time = 30 sec *)
  begin
    writeln('Oscilloskopet triggar inte.');
    OK := false;
  end;
end;


(*****************************************************************************)


FUNCTION READVALUE: REAL;

var    answer      :  array [1..30] of char;
       i,k,
       exponent    :  integer;
       ch          :  char;
       value,
       weight      :  real;

begin
  k := 1;
  exponent := 0;
  repeat
    repeat
      ch := read_char;
    until ch <> chr(0);
    answer[k] := ch;
    k := k + 1;
  until (ch = chr(13)) or (k = 30);
  writeln('All characters are read.');
  WRITELN ('SENT FROM SCOPE   ');
  I := 1;
  REPEAT
    WRITE(' ',ANSWER[I]);
    I := I + 1;
```

```
 UNTIL ANSWER [I] = CHR(13);
 k := 4;                                    (* Begin search at FIFTH position *)
 repeat
   k := k + 1;
 until (answer[k] in ['0'..'9']) or (k = 30);
 if k <> 30 then
 begin
   value := 0;
   while answer[k] in ['0'..'9'] do
   begin
     value := 10*value + ord(answer[k]) - ord('0');
     k := k + 1;
   end;
   if answer[k] = '.' then
   begin
     k := k + 1;
     weight := 1;
     while answer[k] in ['0'..'9'] do
     begin
       weight := 0.1*weight;
       value := value + weight*(ord(answer[k]) - ord('0'));
       k := k + 1;
     end;
   end;
   if answer[k] = 'E' then
   begin
     k := k + 1;
     if answer[k] = '-' then
       exponent := - (ord(answer[k+1]) - ord('0'))
     else if answer[k] = '+' then
       exponent := ord(answer[k+1]) - ord('0')
     else
       exponent := ord(answer[k]) - ord('0');
   end;
   value := value * exp(exponent * ln(10));
   readvalue := value;
  end
  else
    readvalue := 0.0;
end;



(*****************************************************************************)


PROCEDURE FINDPOSITIONS;

(* Note that when scope is in dual channel mode, the 1024/4096 points
are chared between the channels. Consequently, each 'integervector' will
contain 512/2048 integers, BUT the scope will nevertheless report posi-
tions from 2 to 1048/4096. No odd numbers are reported. Not to be confused
carefully notice the use of 'index' numbers and 'position' numbers. *)


const  downinterval     =  1;
       upinterval       =  1;
       downsteps        =  5;
       upsteps          =  2;


var    newpoint,
       ready          :  boolean;
       index,
```

```
      steps          :  integer;
      i              :  integer;

begin
  index := 0;
  steps := 0;
  newpoint := false;
  ready := false;
  repeat
    repeat
      if voltageint[index+downinterval] < voltageint[index] then
      begin
        steps := steps + 1;
        index := index + downinterval;
      end
      else
      begin
        newpoint := true;
        index := index - steps*downinterval + 1;
        steps := 0;
      end;
    until (steps = downsteps) or (newpoint);
    if steps = downsteps then
    begin
      startindex := index - (downsteps-1)*downinterval - 1;
      write(aux,'CURSOR, SELECT:CURS1, TARGET:ACQ, POSITION:',2*startindex,chr(13));
      delay(100);
      ready := true;
    end;
  until (ready) or (index >= 400);
  OK := index < 390;
  if OK then
  begin
    WRITELN('OUT OF FIRST OUTER REPEAT. POS = ',2*startindex);


    steps := 0;
    newpoint := false;
    ready := false;
    index:=startindex;
    repeat
      repeat
        if voltageint[index+upinterval] >= voltageint[index] then
        begin
          steps := steps + 1;
          index := index + upinterval;
        end
        else
        begin
          newpoint := true;
          index := index - steps*upinterval + 1;
          steps := 0;
        end;
      until (steps = upsteps) or (newpoint);
      if steps = upsteps then
      begin
        flankindex := index-(upsteps-1)*upinterval - 1;
        flankpos := flankindex * 2;
        write(aux,'CURSOR, SELECT:CURS2, TARGET:ACQ, POSITION:',flankpos,chr(13));
        delay(100);
        ready := true;
      end;
    until (ready) or (index >= 500);
    WRITELN('OUT OF SECOND OUTER REPEAT. POSITION = ',flankpos);
```

```
    OK := index < 490;
  end;
end;

(*****************************************************************************)



PROCEDURE SMOOTHCURRENT(startindex,stopindex:integer); (* Capacitive system *)

const  tolerance  =  10;

var   index,
      flipout,
      goodagain   :  integer;
      i, k        :  integer;
      oldvalue    :  integer;
      diff        :  real;

begin
  index := startindex;
  repeat
    repeat
      oldvalue := currentint[index];
      index := index + 1;
    until (abs(currentint[index] - oldvalue) > tolerance) or (index = stopindex);
    flipout := index;
    i := 0;
    if index <> stopindex then
      repeat
        oldvalue := currentint[index];
        index := index + 1;
        if abs(currentint[index] - oldvalue) <= tolerance then
          i := i + 1
        else
          i := 0;
      until (i = 3) or (index = stopindex) or (index - flipout >= 20);
    if i = 3 then
    begin
      goodagain := index - 3;
      diff := (currentint[goodagain] - currentint[(flipout-1)])/(goodagain-flipout+1);
      for k := flipout to (goodagain-1) do
        currentint[k] := currentint[(flipout-1)] + round(diff*(k-flipout+1));
    end;
  until (index >= stopindex) or (index - flipout >= 20);
  OK := (index >= stopindex);
  if not OK then
    for i := 100 to stopindex do
      write(currentint[i],' ');
end;


(*****************************************************************************)



PROCEDURE CALC;

var     i,j,k          : integer;
```

```
        voltdiv1,
        voltdiv2     : real;
        coeff        : real;
        trigpos,
        trigindex    : integer;
        timeincr     : real;
        thispost     : posttype;
        channel      : integer;
        dummy        : real;
        correction   : integer;

begin
  write(aux,'CH1? VOLTS',CHR(13));
  voltdiv1 := readvalue;
  write(aux,'CH2? VOLTS',CHR(13));
  voltdiv2 := readvalue;
  coeff := (voltdiv1 * voltdiv2 * probeattenuation) / (25 * 25 * resistance);
  k := 0;
  correction := round(offset*25/voltdiv1/probeattenuation);
  for i := 1 to arraylength do
    voltageint[i] := voltageint[i] + correction;
  i := flankindex;
  for j := 1 to flankindex do
    energyarray[j] := 0;
  repeat
    powerarray[i] := (voltageint[i] * currentint[i] * coeff);
    if powerarray[i] < 0 then
    begin
      powerarray[i] := 0;
      k := k + 1;
    end;
    energyarray[i] := energyarray[i-1] + powerarray[i]*sampleinterval;
    i := i + 1;
until (i = arraylength);

  with thispost do
  begin
    nr := spark;
    deltat := seconds;
    deltav := voltage;
    firstpart := energyarray[flankindex+5];
    total := energyarray[arraylength-4];
  end;

  write(resultfile,thispost);

  with thispost do
    writeln(nr:2,'  ',deltat:8,'  ',deltav:10,'  ',firstpart:10,'  ',total:10,'  DIFF: ',

  writeln('Number of negative powers=',k);
end;



(*****************************************************************************)



PROCEDURE INIT;

begin
  auxinptr := ofs(read_char);
  auxoutptr := ofs(send_char);
  assign(resultfile,'RESULT.DAT');
```

```
  rewrite(resultfile);
  for i := 1 to curvelength do
    store[i] := chr(0);
  badcurves := 0;
  write(aux,chr(13));
  delay(100);
  write(aux,'REMOTE ON',CHR(13));
  delay(100);
  write(aux,'DATA, SOURCE:ACQ,TARGET:REF1,ENCDG:ASCII',chr(13));
  delay(100);
  write(aux,'WFM ENCDG:ASCII');
  repeat
    delay(40);
    write(aux,'WFM? XINCR',chr(13));
    sampleinterval := readvalue;
    writeln('Sampleinterval is read to ',sampleinterval);
  until (sampleinterval > 1.0E-10) and (sampleinterval < 1.0);
  writeln('Program assumes capacitive system.');
end;


(*****************************************************************************)

(*************** S T A R T  O F  M A I N  P R O G R A M ***************)

(*****************************************************************************)


begin
  init;
  spark := 0;
  repeat
    spark := spark + 1;
    OK := true;
(*    waitforspark;  *)
    write(#7);              (*  Beep  *)
    write(#7);
    writeln('Ge ny gnista och tryck ENTER.');
    readln(ch);
    if OK then
    for channel := 1 to 2 do
    begin
      setchannel(channel);
      write(aux,'DATA? CHA',CHR(13));
      READ_IN;
      k := 1;
      repeat
        write(store[k],' ');
        k := k + 1;
      until (store[k] = chr(13)) or (k = 30);
      curvein;
      case channel of
        1 : begin
              transform(store, voltageint);
              OK := true;
              findpositions;
              delay(400);
              if not OK then
              begin
                writeln('The flanks of the voltagecurve are not found');
                badcurves := badcurves + 1;
              end;
              write(aux,'DELTAT?,VALUE',chr(13));
              seconds:=readvalue;
```

```
            write(aux,'DELTAT?, VALUE',CHR(13));
            seconds := readvalue;
            WRITELN('TIME IS READ TO ',seconds);
            write(aux,'DELTAV?, VALUE',CHR(13));
            voltage := readvalue*probeattenuation;
            WRITELN('VOLTAGE IS READ TO ',voltage);
          end;
      2 : if OK then
          begin
            transform (store, currentint);
            smoothcurrent((flankindex-10),arraylength);
            if OK then
              calc
            else
            begin
              badcurves := badcurves + 1;
              writeln('Curve too noisy for acceptance!');
            end;
          end;
    end;
  end;
  writeln('Number of generated sparks so far = ',spark);
until (spark - badcurves) = sparklimit;
writeln('PROGRAM TERMINATED.');
writeln('NUMBER OF ACCEPTED CURVES     = ', spark - badcurves:3);
writeln('NUMBER OF NOT ACCEPTED CURVES = ', badcurves:3);
end.
```

```
  rewrite(resultfile);
  for i := 1 to curvelength do
    store[i] := chr(0);
  badcurves := 0;
  write(aux,chr(13));
  delay(100);
  write(aux,'REMOTE ON',CHR(13));
  delay(100);
  write(aux,'DATA, SOURCE:ACQ,TARGET:REF1,ENCDG:ASCII',chr(13));
  delay(100);
  write(aux,'WFM ENCDG:ASCII');
  repeat
    delay(40);
    write(aux,'WFM? XINCR',chr(13));
    sampleinterval := readvalue;
    writeln('Sampleinterval is read to ',sampleinterval);
  until (sampleinterval > 1.0E-10) and (sampleinterval < 1.0);
  writeln('Program assumes capacitive system.');
end;


(*****************************************************************************)

(*************** S T A R T  O F  M A I N  P R O G R A M ***************)

(*****************************************************************************)


begin
  init;
  spark := 0;
  repeat
    spark := spark + 1;
    OK := true;
(*    waitforspark;  *)
    write(#7);              (*  Beep  *)
    write(#7);
    writeln('Ge ny gnista och tryck ENTER.');
    readln(ch);
    if OK then
    for channel := 1 to 2 do
    begin
      setchannel(channel);
      write(aux,'DATA? CHA',CHR(13));
      READ_IN;
      k := 1;
      repeat
        write(store[k],' ');
        k := k + 1;
      until (store[k] = chr(13)) or (k = 30);
      curvein;
      case channel of
        1 : begin
              transform(store, voltageint);
              OK := true;
              findpositions;
              delay(400);
              if not OK then
              begin
                writeln('The flanks of the voltagecurve are not found');
                badcurves := badcurves + 1;
              end;
              write(aux,'DELTAT?,VALUE',chr(13));
              seconds:=readvalue;
```

```
            write(aux,'DELTAT?, VALUE',CHR(13));
            seconds := readvalue;
            WRITELN('TIME IS READ TO ',seconds);
            write(aux,'DELTAV?, VALUE',CHR(13));
            voltage := readvalue*probeattenuation;
            WRITELN('VOLTAGE IS READ TO ',voltage);
          end;
      2 : if OK then
          begin
            transform (store, currentint);
            smoothcurrent((flankindex-10),arraylength);
            if OK then
              calc
            else
            begin
              badcurves := badcurves + 1;
              writeln('Curve too noisy for acceptance!');
            end;
          end;
    end;
  end;
  writeln('Number of generated sparks so far = ',spark);
until (spark - badcurves) = sparklimit;
writeln('PROGRAM TERMINATED.');
writeln('NUMBER OF ACCEPTED CURVES     = ', spark - badcurves:3);
writeln('NUMBER OF NOT ACCEPTED CURVES = ', badcurves:3);
end.
```