RESCIENCE C

Replication / Physics

[Re] Reproduction of Step width enhancement in a pulse-driven Josephson junction

Sabino Maggi[1, ID]

[1]National Research Council, Institute of Atmospheric Pollution Research, CNR-IIA, Bari, Italy

Edited by

(Editor)

Received

01 March 2020

Published

–

DOI

–

Abstract When I read about the Ten Years Challenge, it was a matter of seconds to decide to take the plunge and partecipate to the contest with the replication of a work of a quarter of century ago. At first, I was fairly dubious to be able to replicate the results of that work, but in the end the effort resulted much easier than I initially thought. The paper describes the tools and trick used for the replication and the lessons learned from this effort. Among them, good documentation is paramount. Likewise, equally important is the use of tools and languages that do not fall into oblivion after one or two iterations.

Introduction

[++ Simulate the junction behaviour... ++]

[++ mcphase calculates both the IV curve and tha phase path (?) of the junction (outside the scope of the present paper) but only for a single value of alpha, f ++]

A Josephson junction is a quantum mechanical device composed of two superconducting electrodes separated by a weak link [1]. For currents lower than a critical value I_C, the device behaves as an uninterrupted superconductor and coupled electrons (Cooper pairs) can cross the weak link without any voltage drop. [ALT+++ Coupled electrons

Correspondence should be addressed to ()

The authors have declared that no competing interests exists.

(Cooper pairs) can cross the weak link without any voltage drop, ~~at least until the current reaches a critical value I_c (dc Josephson effect)...+++~~] When the current is further increased, single electrons originated by the breaking up of Cooper pairs begin to traverse the weak link. The potential difference V between the two superconducting films becomes $\neq 0$ and a state is reached where the junction behaves as a resistance.

Besides its current practical applications, the Josephson junction is important from a physical point of view because it has been the first device showing quantum mechanical effects on a macroscopic scale.

In modern Josephson junctions the weak link usually has the form of a thin insulating tunnel barrier (SIS junction) [2], a normal metal film (SNS junction) [3] or a physical nanoconstriction (ScS junction) [4, 5]. Josephson junctions have found wide usage in several research fields, for example as building blocks for RSFQ digital electronics or quantum computers [6], or as very sensitive magnetometers (SQUIDs) and radiation detectors [7, 8].

But the most successful application of Josephson junctions is surely in voltage metrology. A microwave radiation of frequency f can phase lock the junction current, producing the so-called Shapiro-steps, i.e., current steps at the quantized voltages V_n,

$$V_n = n\frac{h}{2e}f, \quad n = 1, 2, \ldots \tag{1}$$

where h and e are the Plank constant and electron charge, respectively. This ac Josephson effect is at the basis of the quantum voltage standard that since 1987 replaced the previous voltage standard based on Weston cells.

An important research topic at the beginning of the '90s was related to finding ways to increase the amplitude of the current steps induced by the microwave radiation. In fact, the stability of the lock between the junction ~~phase~~ and the applied microwave bias, and therefore its insensitivity to noise events which might switch the junction from one quantized level to another, which is a crucial problem for voltage standard applications.

is strongly dependent on the step amplitude [9].

To increase the step amplitude, a non-sinusoidal microwave signal may be used. In 1990 Monaco showed that, in the limit of a voltage-biased junction, when two phased microwaves of frequencies f and $2f$ are added together, the rf-induced current steps have normalized amplitudes larger than those predicted for a standard sinusoidal radiation [10]. Experiments on the so-called "biharmonic drive" readily confirmed, at least in part, these conclusions, probably because the samples could not be properly considered as voltage biased [11, 12].

Extending further the idea, Monaco showed that the rf-induced current steps of a voltage-biased junction could become as large as the junction critical current for a microwave radiation composed of a train of periodic delta functions. However, in practice a voltage bias configuration does not properly model a real junction, which usually should be considered as current biased. Further, a delta function wave shape is only theoretical and cannot be reproduced in experiments. This led to the idea to investigate what happened to a current-biased Josephson junction irradiated by a more realistic pulsed microwave signal [13, 14]. And this investigation is the object of the present work.

2 Computational context

A first attempt to solve this problem was made by using an electronic analog simulator [15], that could compute the $I - V$ characteristic of a current-biased junction in the framework of the Stewart-McCumber RSJ junction model [16, 17]. The analog simulator was very fast and simple to use, and pairing it to a Hewlett Packard 7475A pen plotter it could produce in just a few seconds beautiful plots of the $I - V$ characteristics of the junction as a function of the simulated microwave signal.[1] The real problem with this approach was that, even if the electronic simulation was very fast, it took ages to

[1] Unfortunately, after 25 years and two relocations, I could not manage to find pictures of the simulator nor the original HP 7475A plots.

measure by hand from the paper plots the amplitude of the rf-induced steps visible on each $I-V$ curve.

A different approach was necessary and I decided to write a FORTRAN program to solve numerically the nonlinear second-order differential equation that models the junction behaviour [16, 17]. The idea was to calculate the $I-V$ curves of the junction as a function of the pulse amplitude α_{rf} for a given set of junction parameters β, Ω and ρ. To ease comparison of the results for different sets of parameters, normalized units have been used throughout the calculations [16, 17]. Usually, around 100 different $I-V$ characteristics for increasing values of α_{rf} were calculated for a for a given set of junction parameters.

The first versions of the FORTRAN program were compiled under DOS 6.22 with Microsoft Fortran 5.1 and run on what was then a state-of-the-art PC, probably a Compaq Deskpro 486 with a math coprocessor, that was shared among several users of the lab. However, after a few attempts the limitations of a standard PC for such a task soon become evident. A new simulation started automatically on late evenings and took the entire night to be completed. Unfortunately, people still using the PC late in the evening, often inadvertently stopped the background process (it was DOS and Windows 3.11 after all, multitasking was still barely usable) or simply shut the machine down without checking if there was some other running job. At the end, I was able to to complete a single simulation run only every two or three days!

Luckily, after a few weeks of these mostly unsuccessful attempts, a colleague [that was also the sysadmin of his research group,] proposed me to use four Digital workstations running Digital UNIX, or more probably ULTRIX, for my own simulations. The machines were heavily used by his research group during working hours but sat mostly idle overnight. If I could manage to finish my runs before the start of the new work day, I was free to use this idle time for my simulations. My colleague gave me a quick crash course on UNIX and I was ready to go.[2] Porting my FORTRAN program from DOS/Mi-

[2] I should still have my notes somewhere.

crosoft Fortran 5.1 to UNIX Fortran was a snap, and I also quickly managed to learn how to use FTP to transfer the configuration files needed by the simulations and the output files containing the results of the simulation back and forth from the DEC machines to my desktop machine (at that time, I was one of the *happy few* to have a personal computer sitting on my own desk).

The real problem now was how to analyse all this data. A manual analysis like that needed by the electronic simulator was impractical, so I decided to try the recently released Microsoft Visual Basic 1.0 to write a program that took the nightly output files and calculated automatically the size of the rf-induced steps visible on the calculated $I - V$ characteristics as a function amplitude of the rf drive, α_{rf}. Everything worked flawlessly.

So at the beginning of each day I had 4 different sets of files coming from the four DEC machines, three of them were made by keeping β and Ω constant and changing the value of ρ, i.e., the (normalized) width of the pulse signal [++ CHIARIRE CHE OGNI NOTTE VENIVANO CAMBIATI I VALORI DI BETA E OMEGA ++]. The fourth simulation was made with exactly the same parameters but using a standard sinusoidal drive and was used to compare the results obtained with a standard sinusoidal microwave signal with those obtained with progressively shorter pulses. I FTPed these files from the DEC workstation to my desktop computer and I run the Visual Basic program on each data set to quickly get a summary file containing the size of the quantized current steps for the reference sinusoidal drive and for the three simulated junctions subjected to progressively shorter current pulses. The results of months of calculations were summarized in a paper published on the Journal of Applied Physics [13].

3 Digging into code

Finding the original sources of this project was easy and complex at the same time. I am the type of person that likes organisation, and I try keep all my past projects on my main

work computer. So finding the original sources was only a question of finding the right directory where the project was stored. Problems arise when I started to look at the different files. There were several directories, with many files with widely different names and dates, trying to find and order in that chaos seemed at first impossible. Normally I would have found many notebooks full of detailed handwritten notes for that project. Unfortunately, a couple of years ago most of my work notebooks were damaged by a water leak in my basement, and could not be recovered. So the only option was to look at the files one by one. Luckily modern operating system have many often little known tools that can greatly ease the work with files, macOS in particular offers QuickLook which allows to quickly browse through tens and hundreds of files at the touch of the spacebar.

After a thorough inspection of the directories of the project I leaned that: (1) my first attempts with the numerical simulations used the more accurate McDonald-Johnson model [18], and that I switched to the simplified Stewart-McCumber RSJ junction model [16, 17] because it allowed much faster and efficient calculations; (2) file names reflected what the programs actually did; and in the same directory I could have a file ending with a "t" that provided a textual output and the same files ending with a "g" that gave a graphical output. Having many files differing only for a couple of DEFINESs might seem crazy today, when graphical interfaces and ultra-fast and ultra-confortable text editors with access to regular expressions allow to change a file in just a few seconds, but probably at that time it was the fastest, albeit inefficient, way of working with source code; (3) all source files had an header containing detailed notes about the type of program, the compiler, the type of output, and what probably mattered more, the dates of first and last revision of the source file (Fig. 1), and that eased much the analysis of the different versions of the FORTRAN source files; (4) the initial versions of the FORTRAN programs were monolithic, there was only one source file containing the whole code (around 1.000 lines of FORTRAN); only at a later moment good computing practice led me to divide the monolithic code into several source files that were compiled and linked together using

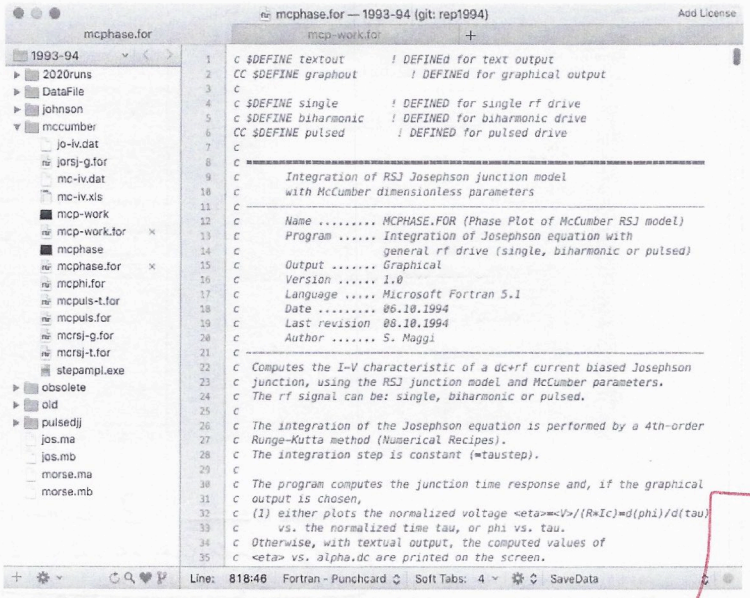

Figure 1. Header of one of the FORTRAN source files.

a Makefile (but I will not delve into it here). A great tool that I used heavily was Meld, an open source tool for all current operating systems that can perform a two- and even a three-way comparison of files and directories. At the beginning it might seem confusing and hard to use, but when one gets the handle of it it becomes and invaluable tool to analyse computer code. Using Meld I quickly realized that the two main source files were mcphase.for and mcp-work.for (Fig. 2), both located in the mccumber directory.

The former performed a simulation for a single value of α_{rf}, so it used a DOS batch file to perform simulations for several values of α_{rf}, renaming the output file containing the results of each simulation, iv.out, using a consistent naming scheme. This approach was typical of DOS and was also very inefficient and error prone, since each day it required to write a long series of .dat input files that differed only by the value of α_{rf}, updating correspondingly the long DOS batch file that controlled the night calculation (Fig. 3). On the other hand, mcp-work.for automatically cycled across a predefined set of values of α_{rf}, producing a different output file for each value of α_{rf}. It was clear that this was the source code ported to the DEC workstations. Unfortunately, I could not find the

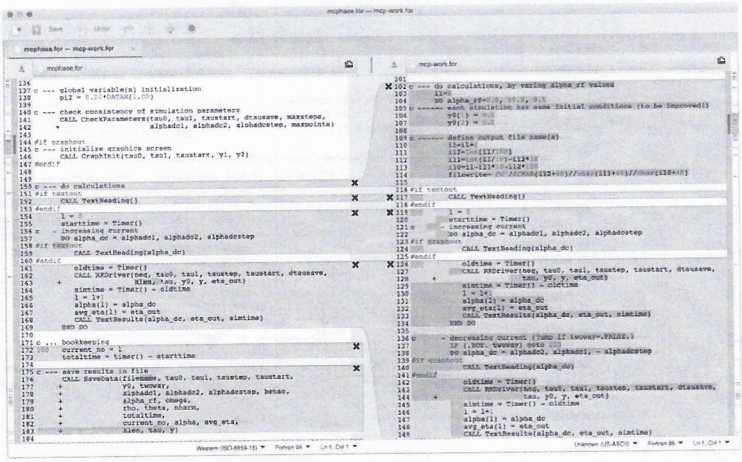

Figure 2. ~~Meld...~~

source file used o these machine, probably I ported my DOS/Microsoft FORTRAN 5.1 to ~~Digital~~ UNIX working directly on the workstations, and I never thought to copy back these sources since they could not work on my PCs. Luckily, FORTRAN is very stable ~~an it was very easy to port~~ again both mcphase.for and mcp-work.for so that it worked on a modern machine.

As for the Microsoft Visual Basic 1.0 code, there were only two versions of the Visual Basic analysis programs and the differences between them ~~seem~~ minimal, so I decided to stick with the initial version.

4 Porting Microsoft FORTRAN to UNIX

Porting mcphase.for and mcp-work.for to the XXI century so that they could ~~be compiled~~ with a modern ~~gfortran~~ compiler was very easy, thanks to the stability of the language across different versions and platforms. Essentialy only a few tweaks to the source code were needed.

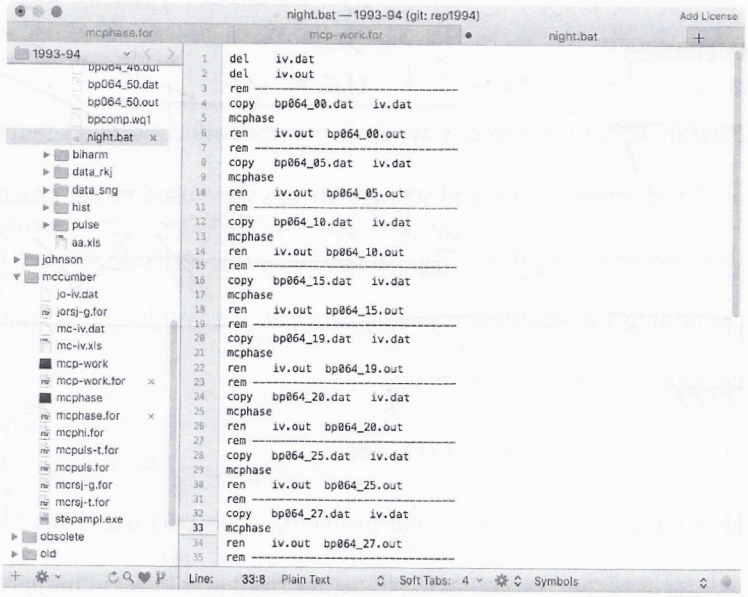

Figure 3. Batch file.

4.1 Preprocessor directives

For reasons that go beyond my understanding Microsoft Fortran 5.1 did not use standard cpp or fpp preprocessors directives (the latter is the de facto standard Fortran preprocessor) [19], but used a a slightly different syntax (Fig. 4). Luckily, all was needed was to comment out all the $DEFINE lines in the header section and to replace throughout the code Microsoft Fortran 5.1 DEFINE blocks with standard cpp blocks (Fig. 4). Now the right directives are chosen at compile-time, by running for example the following command,

```
$ gfortran -cpp -Dtextout -Dsingle -o mcp-work mcp-work.for
```

to run the cpp preprocessor before compilation of mcp-work.for with gfortran, selecting code that produced textual output (-Dtextout) and performed calculations with the the single (sinusoidal) rf drive (-Dsingle).

4.2 Filenames

Fortran 77 did not support dynamic memory allocation and required programmers to use fixed-length arrays and strings. Strings were used rather sparingly in FORTRAN, so that was not a big deal. The exception in my code was for the name of the files containing the simulation parameters and the results of the calculations, which was defined as a 50-byte long string,

```
CHARACTER*50    filename
```

Under DOS that was not a real problem, since DOS truncates file names to 8 characters (plus three characters for the extension) and excess characters were ignored, but under UNIX file names have no real limitations,[3] and having these 50 characters-long filenames, mostly composed by spaces, was ugly and, what matters most, complicated file management, in particular when using a command line interface. The solution is simple: FORTRAN now has the TRIM function that removes all trailing blank characters from a string, so whenever a file is opened for reading or writing, it is sufficient to apply TRIM() to the filename variable, as shown here

```
OPEN (UNIT = 10, FILE = TRIM(filename)//'.dat', STATUS = 'OLD')
```

4.3 Edit descriptors

Microsoft Fortran 5.1 used the backslash \ edit descriptor to prevent the addition of a line break at the end of a WRITE instruction.[4] Modern Fortran compilers do not support this peculiar edit descriptor and return an error, unexpected element '\' in format string. To avoid this error, it is sufficient to remove the backslash edit descriptor from all WRITE instructions that include it.

[3] 255 characters for the name of the file and 4096 characters for the path

[4] http://userweb.eng.gla.ac.uk/peter.smart/com/981124.htm

4.4 Date and time

Microsoft Fortran 5.1 has two separate intrinsic subroutines to return the current date and time. In particular,

```
CALL GETDAT(iyr, imon, iday)
```

saved the date in the two-byte integer variables textsfiyr, textsfimon and textsfiday, while

```
CALL GETTIM(ihr, imin, imin, i100th)
```

did the same for the current time, saving the return values in the integer variables textsfihr, textsfimin, textsfimin and textsfi100th. The meaning of each returned variable should be self-explanatory. Modern Fortran 77 supports the single subroutine DATE_AND_TIME

```
CALL DATE_AND_TIME([DATE, TIME, ZONE, VALUES])
```

where all arguments are optional and can be specified by their dummy name (i.e., how Fortran calls the keyword arguments of a function call). In particular, DATE, TIME and ZONE are character variables, while VALUES is a one-dimensional array of integers containing 8 elements, where VALUES(1:3) corresponds to the year, month and day of the month, VALUES(4) is the time difference (in minutes) with UTC, and VALUES(5:8) are the hour, minute, second and milliseconds, respectively.

To minimize changes to the original source code, the original subroutine calls

```
INTEGER*2       iyr, imon, iday
INTEGER*2       ihr, imin, isec, dummy
...
CALL GETDAT(iyr, imon, iday)
CALL GETTIM(ihr, imin, isec, dummy)
```

is translated to

```
character*8     date
character*10    time
character*5     zone
```

```
integer        values(8)
...
call date_and_time(date, time, zone, values)


iyr  = values(1)
imon = values(2)
iday = values(3)
ihr  = values(5)
imin = values(6)
isec = values(7)
```

4.5 Compilation with gfortran

All these fixes were enough to compile mcphase.for correctly with textsfgfortran. As for the other source file, mcp-work.for, it basically required the same corrections, with one rather strange addition. As noted above, mcp-work.for calculates the $I - V$ characteristic of the simulated junction for several different values of $\alpha_r f$, producing a different output file for each $I - V$ curve. The code to define the output file names was very convoluted,

```
c --- do calculations, by varing alpha_rf values
      il=0
      DO alpha_rf=0.0, 50.0, 0.5
      ...
c ------ define output file name(s)
         il=il+1
         il2=Int(il/100)
         il1=int(il/10)-il2*10
         il0=il-il1*10-il2*100
         filewrite='PU'//CHAR(il2+47)//char(il1+47)//char(il0+47)
```

```
...
END DO          ! repeat alpha_rf DO cycle
```

where il is a counter of the cycle number and il2, il1 and il0 are integers containing the hundreds, tens and units digits of il. These integers are converted to the corresponding ASCII characters with the function CHAR and the name of the output file filewrite is built by concatenating the three ASCII characters (plus a trailing constant string) with the (//) operator, where textsfASCII code 48 corresponds to the 0 symbol and textsfASCII code 57 corresponds to 9.

I have no idea why I decided to use such a complex way to name the output files, but nevertheless it did not work under gfortran and prevented proper compilation of the code. After a little inspection it is clear that the problem lies in the number 47 added to three integer variables before converting them to ASCII characters, that should be replaced by 48, so that the line defining the filewrite variable becomes

```
filewrite='PU'//CHAR(il2+48)//char(il1+48)//char(il0+48)
```

what is really puzzling is why Microsoft Fortran required instead a 47 to convert each integer to the correct ASCII character.

5 Visual Basic code

No attempt was made to make the original Visual Basic 1.0 code run a modern computer. Visual Basic is a dead language and long since has been replaced by Visual Basic .NET, which shares only the name with its forefather. From the beginning the only viable option was to use emulation. For this task I preferred to use Parallels Desktop,[5] which runs only on macOS, but users of Windows or Linux can easily choose the open source VirtualBox package instead.[6] Installing Visual Basic 1.0 in the emulator required several preliminary steps. I had to

[5] https://www.parallels.com/

[6] https://www.virtualbox.org/

install in sequence DOS 6.22, Windows 3.11 and lastly Visual Basic 1.0. While I was at it, and although I had already solved the Fortran part of the job, I decided to install also Microsoft Fortran 5.1 for DOS, to try to reproduce as much as possible the original development environment.

All the software was downloaded from the WinWorld web site,[7], which is an invaluable resource for recovering old software packages. The legitimacy of installing proprietary software in an emulator might be questionable, even after so many years, but in any case at the time I had regular licenses for all the above mentioned software, so I believe to be at least morally to be authorized to use those packages.

Installing the various software packages is very close to how it was done at the time: one has to insert in sequence the several floppy disks on which the package was distributed, with the only difference that floppy disks now are virtual file images and that swapping floppy disks is not done mechanically but requires to select a menu option in the emulator. [++ resolution 640x480 ++]

Another problem to be dealt with was how to transfer the source and data files to the emulated DOS/Windows system. I soon excluded the possibility of try to make Windows 3.11 communicate with the host OS through the network [++ by setting up Windows 3.11 networking in the emulator ++]. I don't even know if it can be done, but it seemed to me too much of a hassle for this simple project. Much better is to use again a virtual floppy disk, that should be mounted alternatively (end exclusively) on macOS or on Windows 3.11. To create a virtual floppy disk data.img it is sufficient to issue the command dd in the macOS Terminal,

```
$ dd if=/dev/zero of=data.img bs=1440k count=1
```

and then run Windows 3.11 in the emulator, mount the virtual floppy disk and format it in the MS-DOS FAT format. Using this virtual floppy disk I copied the source code of mcphase.for and mcp-work.for and the mc-iv.dat configuration file, transferring them in the

[7] https://winworldpc.com

same directories used originally.

[++ how did I find where these dirs were located??? ++]

[++ sorgente stepampl ++] [++ già compilato, VB non è strettamente necessario il sorgente basta il binario ++]

This step completed the preparation stage of the development environment, now it was time to test how all this behaved.

6 Running the programs

To avoid cluttering the mccumber/ directory containing the source files with the output data files produced by the simulations, before trying to run the two Fortran programs I create in the main project folder a new directory, 2020runs/, copying there the mc-iv.dat configuration file needed to start the simulation. [++ before: show layout of the project directory? ++]

Running mcphase.for is easy. In summary, all I had to do was to compile mcphase.for, move to the 2020runs/ directory and run the mcphase executable from there, as summarized below

```
$ gfortran -cpp -Dtextout -Dsingle -o mcphase mcphase.for
$ cd ../2020runs/
$ ../mccumber/mcphase
```

obtaining at the end of the calculations a file mc-iv.out that contains the $I-V$ characteristic and the phase plot (for that, read below) of the junction for the set of parameters defined in mc-iv.dat. The calculation takes just a couple of seconds on a recent Mac, and for $\alpha_{rf} = 0$ produces a standard non-hysteretic $I-V$ curve (Fig. 6a), while for non-zero values of α_{rf} one gets the standard staircase-like $I-V$ curves with the rf-induced current steps (Fig. 6b and Fig. 6c).

To simulate a pulsed rf drive, it is sufficient to compile mcphase.for with the -Dpulsed switch and rerun the simulation (Fig. 7). The Figure shows clearly that with the pulsed rf drive

there are fewer but much larger rf-induced steps and that they can be nearly as wide as the critical current for $\alpha_{rf} = 0.0$ (compare Fig. 7a) and (Fig. 7c).

The second Fortran program, mcp-work.for, is much more interesting. While mcphase.for can calculate the $I - V$ characteristic of the junction for a single value of α_{rf} and for increasing or decreasing values of the normalized current α_{dc}, mcp-work.for automatically sweeps the normalized current α_{dc} in both directions to help visualize better the possible hysteresis of the junction and calculates the $I - V$ characteristics for a range of values of α_{rf}, saving each curve in a separate output file. Unfortunately, the lower and upper limits and the step size of α_{rf} are hardcoded in the Fortran source code ad every change requires a recompilation of mcp-work.for (a minor hassle, as the compilation takes just a couple of seconds on a modern machine).

Running mcp-work.for is very similar to the previous case. First, it is necessary to compile mcp-work.for with the proper compiler switches, then one moves to 2020runs/ directory and runs the mcp-work executable from there. The whole process is summarized below for the standard sinusoidal drive,

```
$ gfortran -cpp -Dtextout -Dsingle -o mcp-work mcp-work.for
$ cd ../2020runs/
$ ../mccumber/mcp-work
```

while the only modification needed to perform calculations using the pulsed drive is to change the -Dsingle switch to -Dpulsed,

```
$ gfortran -cpp -Dtextout -Dpulsed -o mcp-work mcp-work.for
$ cd ../2020runs/
$ ../mccumber/mcp-work
```

Also the names of the output files are hard-coded in the mcp-work.for in the filewrite variable and are conventionally composed by a two-letter prefix ("SI" for the single drive, "BI" for the biharmonic driva and "PU" for the pulsed drive) followed by a tree-digit integer containing the cycle number (see Section 4.5).

On a modern machine the whole calculation with 100 α_{rf} steps takes around $2-4$ minutes and most of the time is spent printing the calculated $I-V$ curves for each value of α_{rf}. Such feedback was necessary at the time of the original calculation, as every new calculated point of the $I-V$ curves appeared on the screen after several minutes, now the results literally flow on the screen at a speed that makes them almost illegible. The output files should be transferred to the Windows 3.11 virtual machine to be processed by stepampl, the Visual Basic application described in Section 5. However Windows 3.11 does not *understand* UNIX line terminators and cannot read the output files without a proper line terminator conversion. This conversion can be easily done in the macOS Terminal with the following command

```
$ for f in $(ls *.out); do sed -i .bak s/$/$'\r'/ $f ; done
```

that converts the line terminators of all the .out output files from the UNIX format containing only a line-feed (LF) to the carriage return followed by a line-feed (CRLF) format used by all versions of Microsoft Windows (slightly different versions of the command can be usually found on internet; however the format used above is POSIX-compliant and should run on all UNIX flavours). The -i switch allows in-place conversion of each files (the original files are saved with the .bak extension and can be easily removed after the conversion).

Now the output files in the proper Windows format can be transferred onto the virtual floppy disk image. When the transfer is done, the floppy disk image must be unmounted from the Desktop of the host operating system and mounted in the emulator. This allows to open the floppy disk in Windows 3.11 so that the output files can be copied in an (preferably) empty directory of Windows 3.11. Now it is the time to run the Visual Basic application, either by running the precompiled stepampl.exe application or by opening the Visual Basic project and running the program form there (Fig. 8). [++ 100 data files max ++] The summary file containing the the size of the quantized current steps for a single simulation are saved in a file with extension .STP (for steps) and can be transferred

to the host operating system via the floppy disk image.

At the time of writing the original paper [13], the whole process had to be repeated each night for a different set of input parameters, or for a different kind of rf drive (single, bi-harmonic or pulsed). As noted above (Section 2), as each night I had four DEC machines available, I could use three of them to simulate in parallel the junction behaviour with the pulsed drive with three different values of the (normalized) width of the pulse signal, ρ, while keeping constant all the other junction parameters, such β and Ω. The only thing that in different in the three simulation is the range over which to change $\alpha_{\rm rf}$, which depends on the value of ρ (shorter pulses require a much larger intensity of the rf-signal to have the same effect on the junction).

The fourth machine was reserved to simulate the junction behaviour with the standard sinusoidal drive, using exactly the same set of junction parameters. Each night a different set of junction parameters was used.

Luckily the original paper contained all the information needed to reproduce the results shown in the figures, without having to repeat the whole analysis from scratch. The parameters used in all runs were: hysteresis parameter $\beta = 0.01$, frequency of the sinusoidal or pulsed rf signal $\Omega = 0.45$, normalized current bias ranging between $\alpha_{\rm dc} = -5.0$ and $\alpha_{\rm dc} = 5.0$, integration time $\tau = 500$, step $\Delta\tau = 0.01$. For the simulation with the sinusoidal drive, the amplitude of the rf signal $\alpha_{\rm rf}$ was varied between 0.0 and 5.0, with a step $\Delta\alpha_{\rm rf} = 0.1$. The three simulations with the pulsed drive were done using: (1) pulse width $\rho = 0.250$ and $\alpha_{\rm rf} = 0.0, 0.1, \ldots 10.0$; (2) pulse width $\rho = 0.125$ and $\alpha_{\rm rf} = 0.0, 0.2, \ldots 20.0$; (3) pulse width $\rho = 0.050$ and $\alpha_{\rm rf} = 0.0, 0.5, \ldots 50.0$.

The resulting output and summary files are saved in separate folders in the 2020runs/ directory, named SINGLE/, PULS0250, PULS125, PULS0050 after their DOS counterparts.

To reproduce the first and second figure of [13] I made the only concession to modernity and, instead of trying to recreate them with the original plotting program used originally, probably Origin 2.0,[8] I decided to write a couple of small R scripts that could automate

[8] https://www.originlab.com

the task. The results are shown in Fig. 9 and Fig. 10 and, as expected, are identical to those reported in the first two figures of [13]. Figure 3 of the original paper could be trivially reproduced by plotting the maxima of the curves of Fig. 9 (Δi_n vs. α_{rf} for the four different cases considered here.

Similar considerations can be made for the reproduction of Figure 4, which considers a slightly hysteretic junction with $\beta = 1.0$. In the spirit of this challenge I have chosen to show instead [++ the $I - V$ curves of the simulated junction with $\beta = 1.0$ and ++] the related Δi_n vs. α_{rf} curves for the two most significant cases of rf drive, i.e., the sinusoidal drive and the pulsed drive with $\rho = 0.050$ (Fig. 11).

7 Discussion

If I would start the project today from scratch I think I would handle things quite differently.

1. Use Python instead of Fortran. Fortran, despite its venerable age, is stil an excellent language for scientific programming, but Python is much more comfortable to use, and the availability of excellent libraries such as NumPy and SciPy and of Fortran and C bindings to enhance performance in time-critical sections of code, make it the best tool today for numerical computations.

2. Do not use Visual Basic. First of all I would avoid the use a new language, as was Visual Basic 1.0 at the time. In 1993-94 it would have been very hard to foresee the rapid demise of Microsoft's Visual Basic, exactly as it would have been impossible to foresee the explosive success of Python, but in any case using a language only when it is stable, runs on a wide array of operating systems and is accepted by a wide community of developers is a safe bet. [++ The same can be said today for Apple's Swift, until the language does not support Windows as well as macOS and Linux and it is not used by a large community outside that of Apple developers, I

~~would never use it for a (serious) (research) project].~~

8 Conclusions

Going back to this old paper has been an exceptionally interesting experience and I must thank the organizers of this challenge for the opportunity offered.

However this is not only a nostalgic attitude. The reproducibility or replicability crisis is a serious issue today [20], when many scientific studies are difficult to reproduce or replicate and true science is challenged daily by cheaters that publish papers with fabricated, falsified, or modified data or results.

Being able to go back and redo what has been done in the past assures that...

[++ SUPPLEMENYTARY INFO??? ++]

References

1. A. Barone and G. Paternò. **Physics and Applications of the Josephson Effect**. Wiley, July 1982.
2. M. Gurvitch, M. A. Washington, and H. A. Huggins. "High quality refractory Josephson tunnel junctions utilizing thin aluminum layers." In: **Applied Physics Letters** 42.5 (Mar. 1983), pp. 472–474.
3. S. P. Benz. "Superconductor normal superconductor junctions for programmable voltage standards." In: **Applied Physics Letters** 67.18 (Oct. 1995), pp. 2714–2716.
4. S. A. Cybart, E. Y. Cho, T. J. Wong, B. H. Wehlin, M. K. Ma, C. Huynh, and R. C. Dynes. "Nano Josephson superconducting tunnel junctions in YBa2Cu3O7–δ directly patterned with a focused helium ion beam." In: **Nature Nanotechnology** 10.7 (July 2015), pp. 598–602.
5. N. De Leo, M. Fretto, V. Lacquaniti, C. Cassiago, L. D'Ortenzi, L. Boarino, and S. Maggi. "Thickness Modulated Niobium Nanoconstrictions by Focused Ion Beam and Anodization." In: **IEEE Transactions on Applied Superconductivity** 26.3 (Apr. 2016), pp. 1–5.
6. K. Likharev and V. Semenov. "RSFQ logic/memory family: a new Josephson-junction technology for sub-terahertz-clock-frequency digital systems." In: **IEEE Transactions on Appiled Superconductivity** 1.1 (Mar. 1991), pp. 3–28.

(a)

```
$DEFINE textout        ! DEFINEd for text output
c $DEFINE graphout     ! DEFINEd for graphical output

c $DEFINE single       ! DEFINED for single rf drive
c $DEFINE biharmonic   ! DEFINED for biharmonic drive
$DEFINE pulsed         ! DEFINED for pulsed drive

$if defined (graphout)
    ...
$endif
...
...
$if defined (textout)
    ...
$endif
...
$if defined (pulsed)
    ...
$endif
```

(b)

```
CC $DEFINE textout     ! DEFINEd for text output
c $DEFINE graphout     ! DEFINEd for graphical output

c $DEFINE single       ! DEFINED for single rf drive
c $DEFINE biharmonic   ! DEFINED for biharmonic drive
CC $DEFINE pulsed      ! DEFINED for pulsed drive

#if graphout
    ...
#endif
...
...
#if textout
    ...
#endif
...
#if pulsed
    ...
#endif
```

Figure 4. Preprocessor directives in (a) Microsoft Fortran 5.1, (b) modern cpp preprocessor.

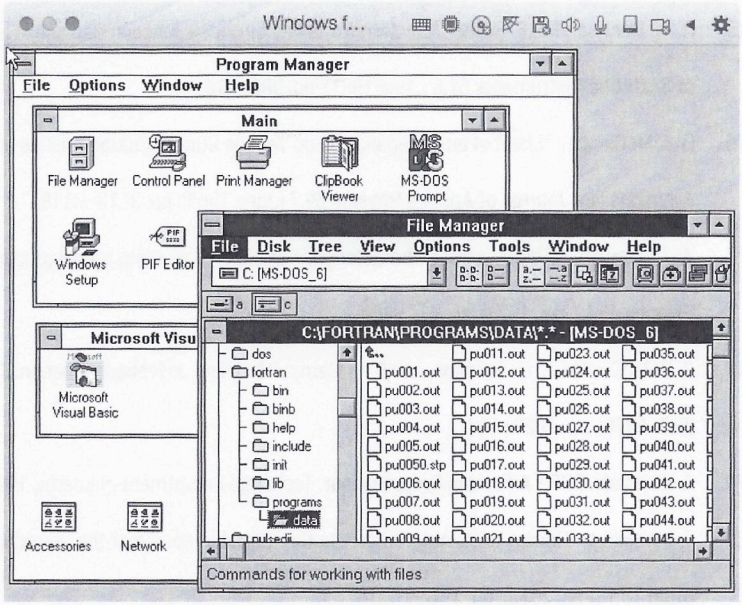

Figure 5. The Windows 3.11 desktop as shown in the Parallels emulator.

7. S. Maggi, N. De Leo, V. Lacquaniti, A. Agostino, R. Gonnelli, and P. Verhoeve. "Nb/Al STJ detectors with sub-nA subgap current." In: **Physica C: Superconductivity and its Applications** 435.1-2 (Mar. 2006), pp. 103–106.
8. C. Granata, A. Vettoliere, R. Russo, M. Fretto, N. D. Leo, E. Enrico, and V. Lacquaniti. "Ultra High Sensitive Niobium NanoSQUID by Focused Ion Beam Sculpting." In: **Journal of Superconductivity and Novel Magnetism** 28.2 (Feb. 2015), pp. 585–589.
9. R. Kautz, C. Hamilton, and F. Lloyd. "Series-array Josephson voltage standards." In: **IEEE Transactions on Magnetics** 23.2 (Mar. 1987), pp. 883–890.
10. R. Monaco. "Enhanced ac Josephson effect." In: **Journal of Applied Physics** 68.2 (July 1990), pp. 679–687.
11. D. Andreone, V. Lacquaniti, and S. Maggi. "Experiments on Josephson Junctions Driven by a Bi-Harmonic RF Source." In: **Nonlinear Superconductive Electronics and Josephson Devices**. Boston, MA: Springer US, 1991, pp. 37–43.
12. D. Andreone, V. Lacquaniti, and S. Maggi. "Numerical and Experimental Results on Josephson Junctions Irradiated by a Biharmonic Drive." In: **Superconducting Devices and Their Applications**. 1992, pp. 399–402.
13. S. Maggi. "Step width enhancement in a pulse driven Josephson junction." In: **Journal of Applied Physics** 79.10 (May 1996), pp. 7860–7863.
14. S. Maggi. "Enhanced phase locking in a Josephson junction driven by current pulses." In: **Journal of Low Temperature Physics** 106.3-4 (Feb. 1997), pp. 399–404.
15. R. W. Henry and D. E. Prober. "Electronic analogs of double junction and single junction SQUIDs." In: **Review of Scientific Instruments** 52.6 (June 1981), pp. 902–914.
16. D. E. McCumber. "Effect of ac Impedance on dc Voltage Current Characteristics of Superconductor Weak Link Junctions." In: **Journal of Applied Physics** 39.7 (June 1968), pp. 3113–3118.
17. W. C. Stewart. "Current voltage characteristics of superconducting tunnel junctions." In: **Journal of Applied Physics** 45.1 (Jan. 1974), pp. 452–456.
18. D. G. McDonald, E. G. Johnson, and R. E. Harris. "Modeling Josephson junctions." In: **Physical Review B** 13.3 (Feb. 1976), pp. 1028–1031.
19. A. Boyanski. **FPP - A Fortran Preprocessor**. Tech. rep. Department of Energy, 1992, pp. 1–7.
20. T. Miyakawa. "No raw data, no science: another possible source of the reproducibility crisis." In: **Molecular Brain** 13.1 (Feb. 2020), pp. 1–6.

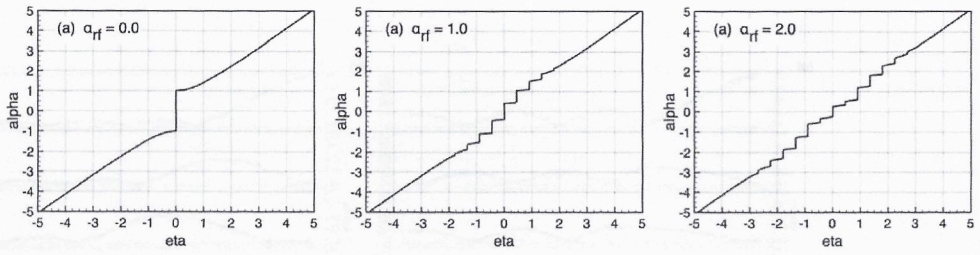

Figure 6. $I - V$ characteristics of a junction with $\beta_c = 0.01$, driven by a sinusoidal rf drive of frequency $\Omega = 0.45$ and (a) $\alpha_{\rm rf} = 0.0$, (b) $\alpha_{\rm rf} = 1.0$ and (c) $\alpha_{\rm rf} = 2.0$. The rf-induced current steps are clearly visible when $\alpha_{\rm rf} > 0$.

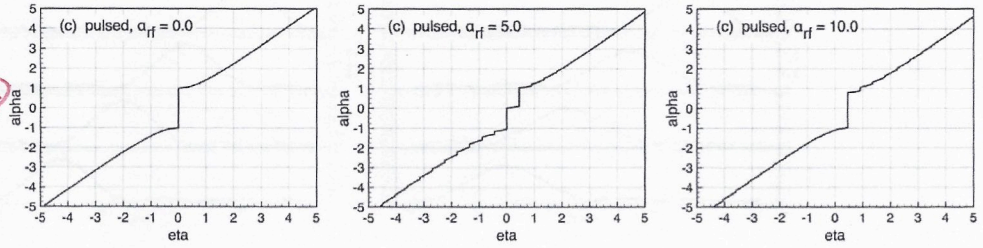

Figure 7. $I-V$ characteristics of a junction with $\beta_c = 0.01$, driven by a pulsed rf drive of frequency $\Omega = 0.45$ for: (a) $\alpha_{\rm rf} = 0.0$, (b) $\alpha_{\rm rf} = 5.0$ and (c) $\alpha_{\rm rf} = 10.0$. For $\alpha_{\rm rf} > 0$ the rf-induced steps are much larger than with the standard sinusoidal rf-drive.

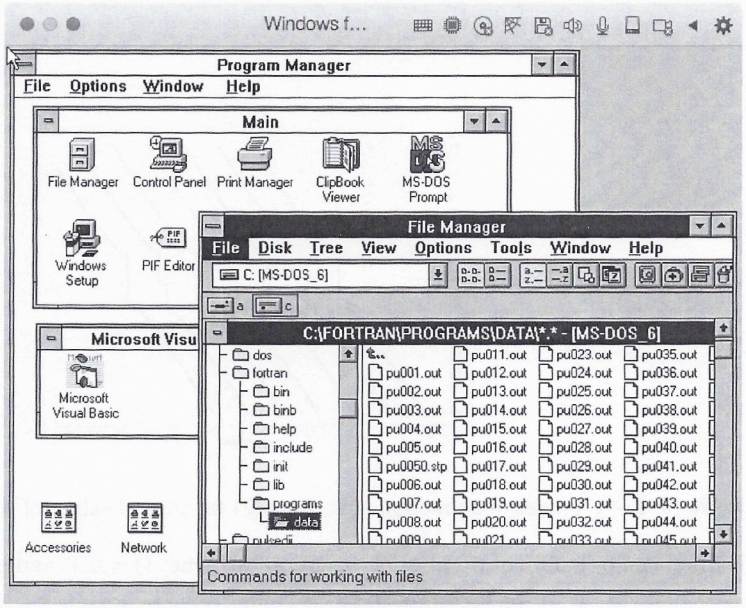

Figure 8. Windows 3.11 desktop showing the program stepampl that has just finished to process the data files.

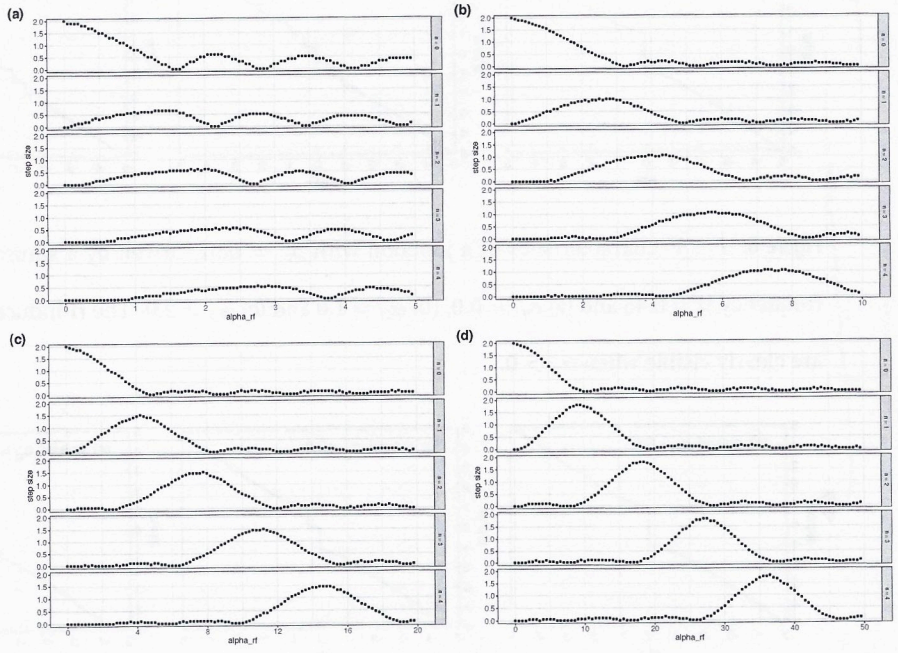

Figure 9. Dependence of the step size Δi_n on the amplitude of $\alpha_{\rm rf}$, for $n = 0...4$. The $n = 0$ step is the normalized total critical current Δi_0 of the junction. The junction parameters are $\beta = 0.01$ and $\Omega = 0.45$; (a) sinusoidal drive, (b) pulsed drive with $\rho = 0.250$, (c) pulsed drive with $\rho = 0.125$, (d) pulsed drive with $\rho = 0.050$.

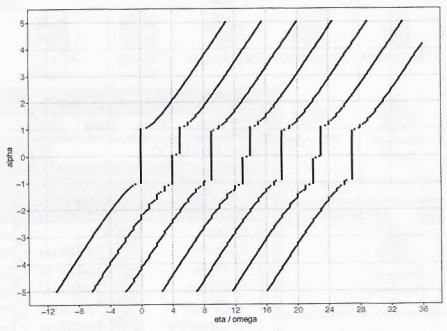

Figure 10. $I - V$ curves of a junction with $\beta = 0.01$ for several values of the rf bias. The junction is irradiated by a train of rf pulses of repetition frequency $\Omega = 0.45$ and pulse duration $\rho = 0.05$. Each curve has been offset by 4Ω. The vertical dotted lines mark the position of the zero-voltage axis for the $I - V$ curve located immediately to the right.

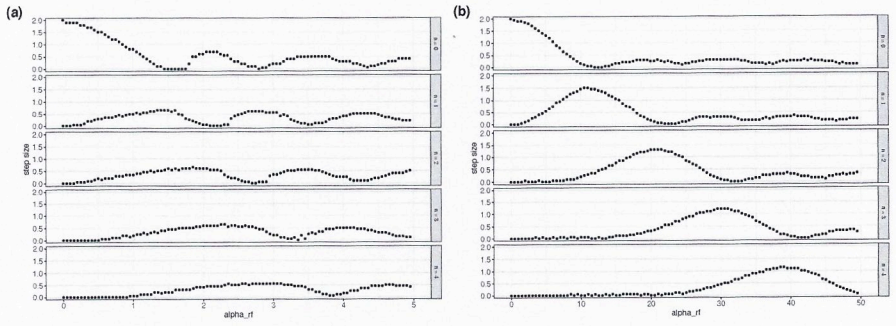

Figure 11. Dependence of the step size Δi_n on the amplitude of α_{rf}, for $n = 0...4$. The $n = 0$ step is the normalized total critical current Δi_0 of the junction. The junction parameters are $\beta = 1.0$ and $\Omega = 0.45$; (a) sinusoidal drive, (b) pulsed drive with $\rho = 0.050$.